College of Saint Benedict and Saint John's University
DigitalCommons@CSB/SJU

Honors Theses, 1963-2015 | Honors Program

1993

Monitoring Shared memory: A Toolset Prototype for the INFORMIX OnLine Administrator

Eric Gronholz
College of Saint Benedict/Saint John's University

Follow this and additional works at: https://digitalcommons.csbsju.edu/honors_theses

 Part of the Computer Sciences Commons

Recommended Citation

Gronholz, Eric, "Monitoring Shared memory: A Toolset Prototype for the INFORMIX OnLine Administrator" (1993). *Honors Theses, 1963-2015*. 343.
https://digitalcommons.csbsju.edu/honors_theses/343

Available by permission of the author. Reproduction or retransmission of this material in any form is prohibited without expressed written permission of the author.

Monitoring Shared Memory:
A Toolset Prototype for the INFORMIX OnLine Administrator

A THESIS
The Honors Program
College of St. Benedict/St. John's University

In Partial Fulfillment
of the Requirements for the Distinction "All College Honors"
and the Degree of Bachelor of Arts
In the Department of Math/Computer Science

by

Eric B. Gronholz
May, 1993

Lynn R. Ziegler, Advisor

Monitoring Shared-Memory
A Toolset Prototype for the INFORMIX OnLine Administrator

Approved by: _____
Lynn R. Ziegler, Associate Professor of Computer Science

Department Readers:

(1) _____
Noreen L. Herzfeld, Associate Professor of Computer Science

(2) _____
J. Andrew Holey, Associate Professor of Computer Science

(3) _____
Michael R. Ross, Computer Science Department Chair

(4) _____
Margaret L. Cook, Honors Program Director

Table of Contents

Abstract	6
Contents of Figures and Listings	7
Acknowledgements	8
Introduction	9
The X Window System	10
INFORMIX OnLine and **tbstat**	15
"Monitor" Program	19
Improvements	29
Conclusion	30
Bibliography	31
Appendices	32

ABSTRACT

The "Monitor" Program was developed to assist system administrators for the INFORMIX OnLine Database Server with their evaluation of shared memory components used on their system. Using the Open Software Foundation (OSF)/Motif Widget set for Xt Intrinsics on the X Window System, this program creates scaled dials and pop up windows to display system information at regular intervals. The program is surprisingly simple to operate and is general enough that with just a few modifications, it can be used to monitor any system information, not just shared memory information for OnLine. This program marks a new way to look at systems analysis, and could be the building block of many powerful programs of the future.

Contents of Figures and Listings

Figure-1	The Client/Server Model	13
Figure-2	Complete X Window System	13
Figure-3	Widgets	15
Figure-4	Two Process Architecture	17
Figure-5	INFORMIX-OnLine Shared Memory	17
Listing-1	Three Processes of Program	20
Listing-2	**tbstat** Output	22
Figure-6	The "Monitor" Program	25
Figure-7	Creation of Child Process	27

Acknowledgements

I would like to thank Joe Foley and the rest of the people at Systems Migration, Inc. for the opportunity to create a product for them.

I would like to thank my friends and family who put up with some unnerving times throughout the course of this project.

I would like to thank the faculty members on my thesis committee for their support throughout this project.

Most of all, I would like to express my deepest thanks to Lynn Ziegler. Not only did he act as an advisor and instructor, but sometimes took on the role of counselor, and certainly became a friend.

Thank you to all of you, for without any of you this project never would have been completed.

Sincerely,

Eric B. Gronholz
May, 1993

Introduction

As the need for faster and more accurate access to information increases, so does the responsibilty of system administrators to keep computer systems and database packages operating efficiently. This responsibility presents administrators with many challenges, but one in particular is their accessibility to information concerning certain shared memory structures. Often the information they need to evaluate how well the system is utilizing its resources is hidden in long and cluttered files. These files can be confusing and extremely time consuming to read. What is needed then, is a program which will show system administrators this information quickly and in an easy to follow format.

"Monitor" is a unique computer application that combines the C programming language with the X Window System to produce a graphical representation of specific shared memory structures utilized by the INFORMIX OnLine Database Server. Through the use of pop-up windows and scaled dials, system information is displayed in a format which makes it easily available to the system administrator. Because structures are independently monitored and displayed, the administrator can isolate potential or current shared memory problems more quickly and easily than normal monitoring allows, thereby facilitating database maintenance and tuning. In short, this tool is useful for evaluating the

system's efficiency and for identifying the kind of activity most commonly executed on the system.

The information displayed by "Monitor" is actually output generated by the INFORMIX OnLine utility **tbstat**. When run at the command line, **tbstat** reads the shared memory structures and reports their contents to the screen at the instant the utility is run. It should be noted that "Monitor" was developed on an SGI (Silicon Graphics) Indigo R3000 machine running the Unix operating system. This system did not have INFORMIX OnLine installed, however, so the **tbstat** utility was not available. To counter the problem of needing this utility's output to develop and demonstrate "Monitor", I ran the utility on a system supporting OnLine and redirected the output to a file. I then wrote my own **tbstat** program which simply lists the file containing actual **tbstat** output to the screen. Hence the program runs as if it were on a system which supports the utility, allowing for demonstrations of the program on *any* X supported system. A more technical explanation of my **tbstat** program will be given when the three processes of "Monitor" are described.

Although "Monitor" was specifically written to display system information for the INFORMIX OnLine Database Server, it is actually a very general program. This statement may sound contradictory, but it is true. "Monitor" resulted from a need to display OnLine system

information in an easy to read fashion. The program's design, however, allows it to display any kind of desired information with relatively few modifications to the display portion of the program. The main work involved in using this program with other system information comes in writing a program which retrieves the system data and puts it in the correct format, not in changing the way the data is presented graphically. This quality should make "Monitor" a very attractive product to all kinds of system administrators, since the same overall design works for many kinds of system monitoring tasks.

The X Window System

To display any kind of system information, "Monitor" uses the X Window System, in particular the OSF/Motif widget set. Douglas A. Young, in his book *The X Window System®: Programming and Applications with Xt, OSF/Motif® Edition*, explains that the X Window System is an industry-standard software system that allows programmers to develop portable graphical user interfaces. This portability means X programs which display windows, text, graphics, etc., can be designed and run on any hardware which supports the X protocol, regardless of that hardware's operating system. In other words, X is device-independent: a program designed on one kind of hardware can be displayed on a different type of hardware without needing to be recompiled or relinked (Young, 1).

The X Window System implements device-independence through the client/server model. This model allows an application to be created and run independently of the process which does the X work (see Figure-1). The X server process handles all input and output devices, and creates graphics, windows, and text as it receives requests for them from the client applications. Once the requests are received and the graphics created, the server sends its work out to the client application where it can be viewed. As Figure-1 shows, the client and the X server need not be on the same machine for the X program to work. The devices need only be linked by a network using the X protocol. This feature is what makes X such a practical user interface development tool.

The flexibility given to clients to create their own user interface does not come without responsibility, however. Since clients can develop everything from the way windows are handled to the shape of push buttons, they are required to account for every detail necessary to create an X application. This requirement means a lot of extra work for the client because developing applications in pure X is a very difficult and tedious process; even the simplest program managing windows would require hundreds of lines of code to accomplish its task. Therefore, various programming groups have developed libraries containing certain X functions which applications can use to do X programming work without having to be concerned with the minute details of X. The most

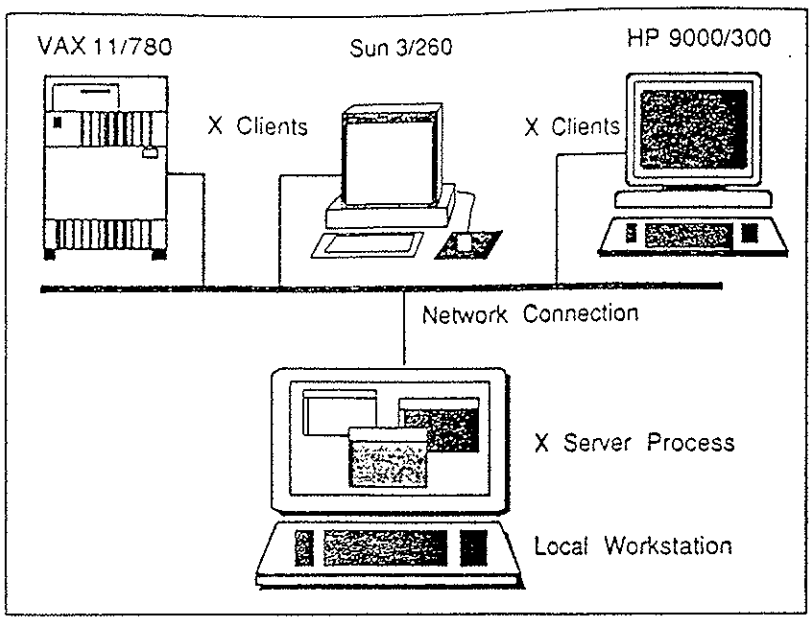


Figure 1. The client-server model.

(Young, 3)

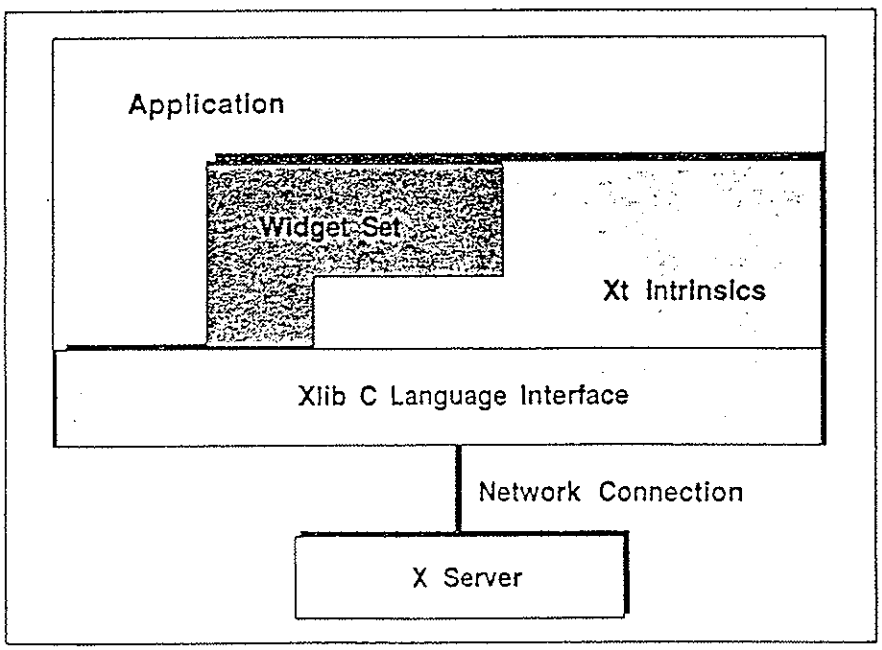


Figure 2. Programmer's view of the complete X Window System.

(Young, 12)

common of these libraries is the C based Xlib library (see Figure-2), which defines an extensive set of functions providing complete access and control over the display, windows, and input devices (Young, 11).

Although these functions are useful in bringing X to a slightly higher level of programming, Xlib is still a very low-level interface. Programming in Xlib, although done much more often than programming in pure X, is also very difficult and tedious. Most X programming is done using some type of standard toolkit which raises the programming to a higher level. "Monitor" uses the X Toolkit, which is made up of two layers: the Xt Intrinsics and one of many sets of functions used to develop graphical objects called widgets, the Open Software Foundation (OSF)/Motif widget set. Figure-2 shows how this toolkit relates to the Xlib interface and the X server. The toolkit, through widget sets, allows programmers to implement user interface components, such as popup windows, push buttons, and menus with relatively few lines of code (see Figure-3).

The OSF/Motif widget set does not directly support the dial widget shown in Figure-3c and used in "Monitor." The dial is actually a widget created by Douglas Young, who has given permission to users of his book to implement and use it as they please. "Monitor" uses the dial widget to display the current status of different shared memory structures, changing the position of the dials' indicators as the different

FIGURE-3 Widgets

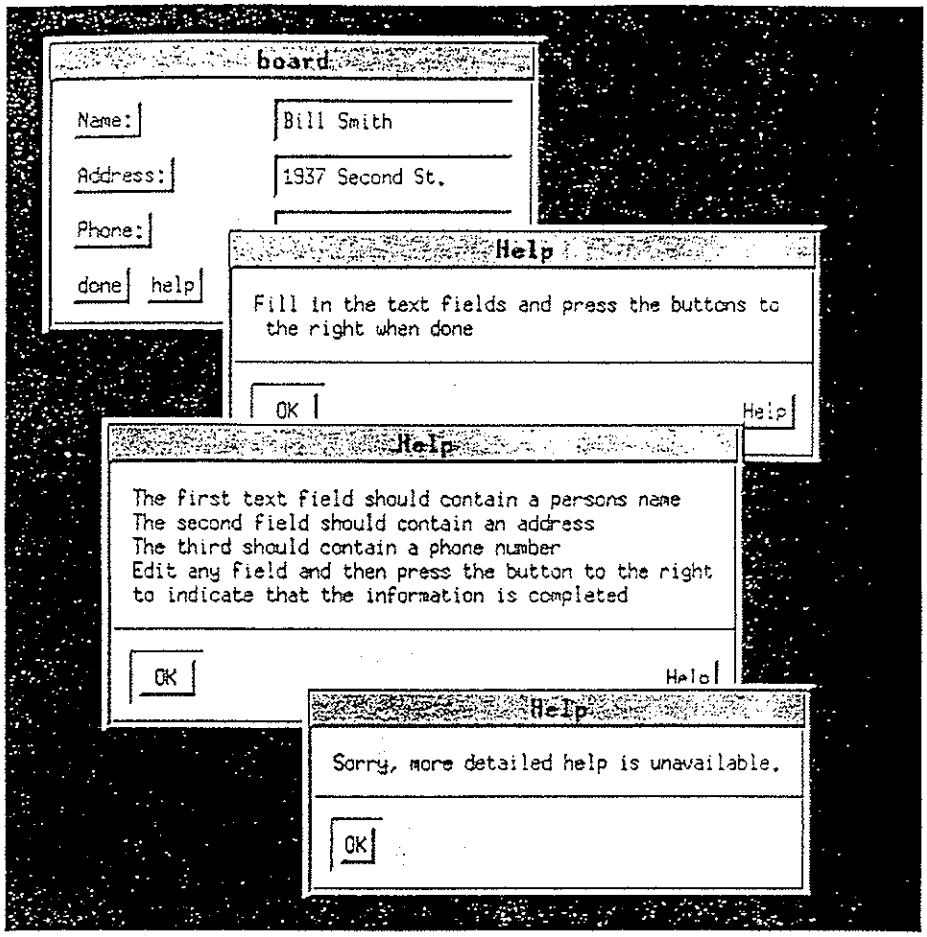


Figure 3a The XmMessageDialog displaying help messages.

(Young, 121)

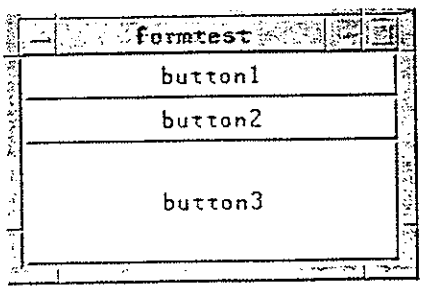


Figure 3b

(Young, 90)

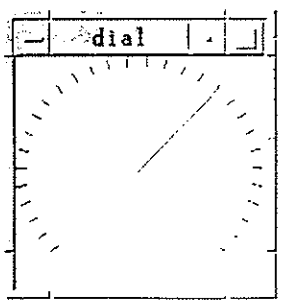


Figure 3c A Dial widget.

(Young, 344)

values change.

INFORMIX OnLine and **tbstat**

INFORMIX OnLine is the system administrating process used to control the INFORMIX relational database. Its architecture is similar to that of the X Window System's in that it uses the client/server model for its access to the database. Figure-4 gives a representation of how this particular client/sever model works. Application programs, usually written in relational database languages like INFORMIX-4GL, make requests to the database server through Unix pipes either to change information in the database or to retrieve something from the database. The database server accesses the information requested and returns the data or a notice that the change to the database failed or succeeded. In either case, the server has sole access to the database, just as the X server process has access to the X code needed to create the requested graphics (INFORMIX, 1002-4).

Shared memory on this server refers to the usage of the same memory segments by more than one OnLine user process, thereby allowing processes to access copies of the same database information. By using copies of the database tables, OnLine runs more efficiently than systems where data is privately stored for each user. For example, database table information can still be stored on multiple disks to reduce data storage limitations, but disk I/O is also reduced because

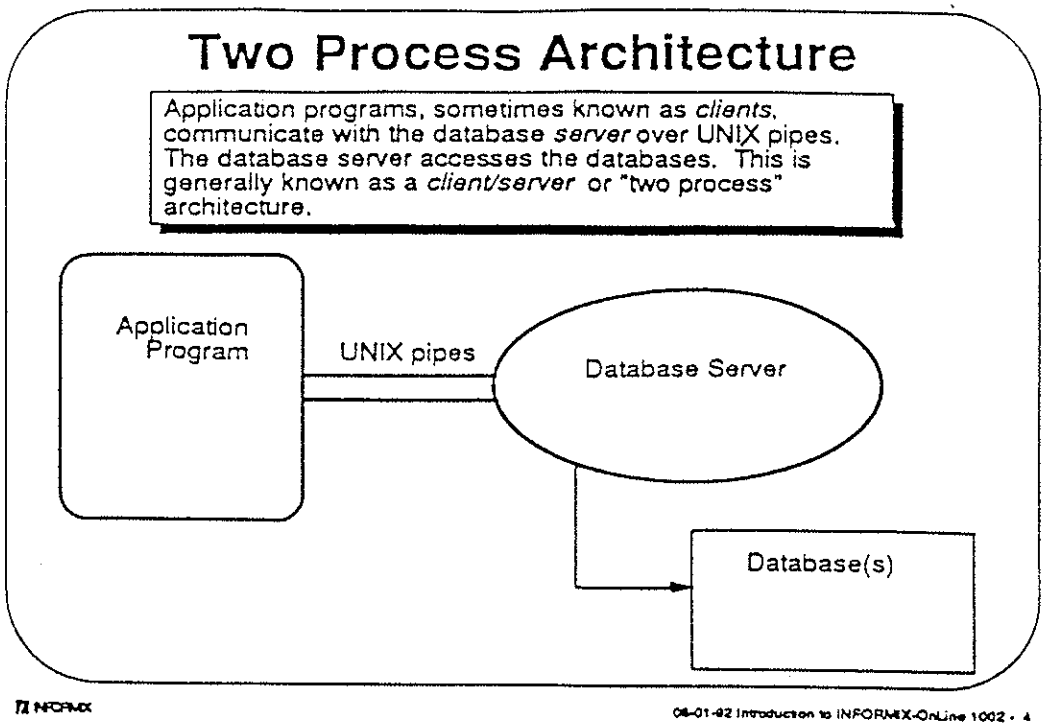
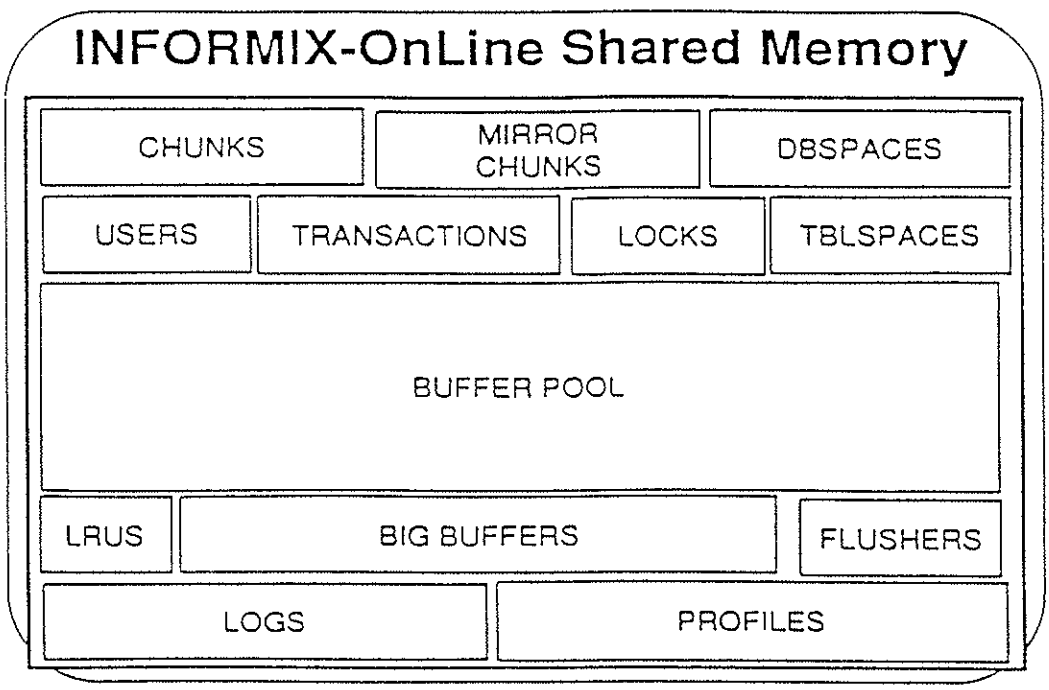


FIGURE 4
(INFORMIX, 1002-4)

FIGURE 5
(INFORMIX, 1715-6)



commonly pooled buffers are flushed on a system-wide basis rather than for each user. As a result, execution time decreases: only one copy of the data is necessary in shared memory and users can access this memory rather than accessing the disk.

In addition to holding copies of database tables, shared memory is also used to maintain system resources for OnLine. OnLine creates various collections of data structures which are allocated in shared memory. OnLine then stores information in these structures that monitors how the database server is being used. Figure-5 lists the major resources of OnLine's shared memory. The specifics on the different components are beyond the scope of this thesis, but system administrators analyze these items to check the efficiency of the system (INFORMIX, 1715-6). As a result, the values of many of the components in Figure-5, along with several others, will be what the dials in "Monitor" will be displaying.

"Monitor" Program

To display the different shared memory structures, "Monitor" utilizes three independent processes: **dial**, **tbstat**, and **alarm**. The brief outline in Listing-1 shows the basic functions of the three processes. The first process, **dial**, acts as the main process for the program. An analogy which might be useful in understanding how the three processes work together is to think of **dial** as a client and the other two processes

as servers: **dial** runs the actual application program while the other two processes do some kind of work at the request of **dial**.

The first request **dial** makes is to the **tbstat** process. Remember that if "Monitor" were installed on a system which supported the INFORMIX OnLine Database Server, this call would be handled by the actual **tbstat** utility installed on that system. Since this program was not implemented on such a system, I developed a process which simulates what the utility does. By writing my own version of this utility, changing "Monitor" to run on an OnLine system becomes trivial. The call to **tbstat** comes from within the function "call_tbstat" in the dial.c program. This function uses the Unix command "system" which allows for the execution of separate programs from within another program. To execute **tbstat** from within **dial**, we use the following line of code:

```
system("tbstat > tbstat.info");
```

What this command says is to execute the **tbstat** utility and redirect its output into a file called "tbstat.info". The only syntax that would need to be changed to run this call on a system supporting OnLine is to add a parameter telling **tbstat** what information to print. Furthermore, since this code appears only once in the "call_tbstat" function, only one line of code needs to be altered. Below is how that line of code will look when the actual utility is being used:

Listing-1

THREE PROCESSES OF PROGRAM

DIAL

Run tbstat and save info in a file.

Collect tbstat info from the file.

Create dials and display info from tbstat file.

If dial hits alarm position, launch subprocess to display the proper alarm. Do not display alarm if currently active.

If help request given, launch subprocess to display proper help window.

Repeat the above events at specified intervals.

TBSTAT

Generate info just like the OnLine tbstat command does. Display a file with correct info to screen.

ALARM

Receives parameter telling it which alarm to print: warning or help. Process is complete when window is killed.

```
system("tbstat -a >tbstat.info");
```

If this parameter is added, no other changes would need to be made for the program to run on an OnLine system because the **tbstat** utility I designed creates the same output as the **tbstat -a** command. Listing-2 shows the format of this output.

The actual **tbstat** command, as I have already explained, reads shared memory structures to create its output. My version of the utility could not do this task however. Instead, "Monitor's" **tbstat** command displays an output file identical to the output generated by executing **tbstat -a** on a system that supports the OnLine Server. To ensure a wide variety of output which covered all boundary cases of the data created by **tbstat**, I made numerous copies of this output file and altered the data in each file in a manner which would properly test the "Monitor" program. Therefore, when **dial** makes the call to **tbstat**, my version simply lists one of these files to standard output, which **dial** redirects to another file.

After **dial** runs **tbstat** and creates the "tbstat.info" file, it continues by reading through this file to get the information which is to be displayed. The information is collected in an array of structures which holds the same kind of information for each shared memory component to be displayed. Appendix A shows all of the components on which **dial**

Listing 2 tbstat output

```
RSAM Version 5.00.UC2  -- On-Line -- Up 10 days 07:14:16 -- 1884 Kbytes

Message Log File: /usr/Informix5/online.log
ted
11:01:09  Checkpoint Completed
11:02:28  Checkpoint Completed
11:03:48  Checkpoint Completed
11:04:39  Checkpoint Completed
11:05:29  Checkpoint Completed
11:06:19  Checkpoint Completed
11:07:09  Checkpoint Completed
11:08:26  Checkpoint Completed
11:09:49  Checkpoint Completed
11:10:48  Checkpoint Completed
11:11:28  Checkpoint Completed
11:12:48  Checkpoint Completed
11:13:28  Checkpoint Completed
11:14:09  Checkpoint Completed
11:15:49  Checkpoint Completed
11:16:58  Checkpoint Completed

RSAM Version 5.00.UC2  -- On-Line -- Up 10 days 01:09:58 -- 1888 Kbytes

Configuration File: /usr/informix5/etc/tbconfig
#*********************************************************************
#
#  INFORMIX SOFTWARE, INC.
#
$Header:
#/26cs1ic1:16:38:91/tbconfig.std        9.3
#  Description: INFORMIX-Online Configuration Parameters
#
#*********************************************************************

# Root Dbspace Configuration
ROOTNAME        rootdbs         # Root dbspace name
ROOTPATH        /dev/rdsk/zd2s2 # Path for device containing root dbspace
ROOTOFFSET      0               # Offset of root dbspace into device (Kbytes)
ROOTSIZE        47000           # Size of root dbspace (Kbytes)

# Disk Mirroring Configuration
MIRROR          1               # Mirroring flag (Yes = 1, No = 0)
MIRRORPATH                      # Path for device containing root dbspace mirror
MIRROROFFSET    0               # Offset into mirror device (Kbytes)

# Physical Log Configuration
PHYSDBS         logdbs          # Name of dbspace that contains physical log
PHYSFILE        2000            # Physical log file size (Kbytes)

# Logical Log Configuration
LOGFILES        4               # Number of logical log files
LOGSIZE         2000            # Size of each logical log file (Kbytes)

# Message Files
MSGPATH         /usr/informix5/online.log # Online message log pathname
CONSOLE         /dev/console    # System console message pathname

# Archive Tape Device
TAPEDEV         /dev/TAPE.CART  # Archive tape device pathname
TAPEBLK         16              # Archive tape block size (Kbytes)
TAPESIZE        61440           # Max amount of data to put on tape (Kbytes)

# Logical Log Backup Tape Device
LTAPEDEV        /dev/null       # Logical log tape device pathname
LTAPEBLK        16              # Logical log tape block size (Kbytes)
LTAPESIZE       61440           # Max amount of data to put on log tape (Kbytes)


# Identification Parameters
SERVERNUM       1               # Unique id associated with this Online instance
DBSERVERNAME    vs              # Unique name of this Online instance

# Shared Memory Parameters
RESIDENT        0               # Forced residency flag (Yes = 1, No = 0)
USERS           20              # Maximum number of concurrent user processes
TRANSACTIONS    20              # Maximum number of concurrent transactions
LOCKS           20000           # Maximum number of locks
BUFFERS         200             # Maximum number of shared memory buffers
TBLSPACES       500             # Maximum number of active tblspaces
CHUNKS          8               # Maximum number of chunks
DBSPACES        8               # Maximum number of dbspaces and blobspaces
PHYSBUFF        32              # Size of physical log buffers (Kbytes)
LOGBUFF         32              # Size of logical log buffers (Kbytes)
LOGSMAX         8               # Maximum number of logical log files
CLEANERS        1               # Number of pagecleaner processes
SHMBASE         0x100000          # Shared memory base address
CKPTINTVL       300             # Checkpoint interval (in seconds)
LRUS            8               # Number of LRU queues
LRU_MAX_DIRTY   60              # LRU modified begin-cleaning limit (percent)
LRU_MIN_DIRTY   50              # LRU modified end-cleaning limit (percent)
LTXHWM          80              # Long TX high-water mark (percent)
LTXEHWM         90              # Long TX exclusive high-water mark (percent)

# Machine- and Product-Specific Parameters
DYNSHMSZ        0               # Dynamic shared memory size (Kbytes)
GTRID_ENT_SZ    32              # Number of bytes to use in GTRID comparison
DEADLOCK_TIMEOUT 60               # Max time to wait for lock in distributed env.
TXTIMEOUT       300             # Transaction timeout for 2-STAR (in seconds)
SPINCNT         0               # No. of times process tries for latch
STAGEBLOB                       # Reserved for INFORMIX-OnLine/Optical

# System Page Size
BUFFSIZE        4096            # Page size (do not change!)

RSAM Version 5.00.UC2  -- On-Line -- Up 10 days 01:26:32 -- 1889 Kbytes

Users
address  flags   pid     user     tty      wait     tout locks nreads  nwrites
401d9c   ------D 277     Informix syscon   0        0    0     1092   448
401c68   ------D 0       Informix syscon   0        0    0     0      0
401c74   ------F 278     Informix          0        0    0     0      3
401db8   ------- 4117    alena    tty05/0A 0        0    1     5      0
401efc   ------- 3160    appa     tty3A/0A 0        0    1     0      0
 5 active, 20 total

Transactions
address  flags  user     locks log begin isolation retrys coordinator
402140   A---- 401d9c   0     0         NOTRANS   0
4021e0   A---- 401d68   0     0         NOTRANS   0
401d9c   A---- 401db8   1     0         COMMIT    0
4030d8   A---- 401efc   1     0         COMMIT    0
 19 active, 20 total

RSAM Version 5.00.UC2  -- On-Line -- Up 10 days 01:35:08 -- 1858 Kbytes

Locks
address  wtlock   owner    lklist   type     tblsnum  rowid    size
40d510   0        401ce0   0           S     1000002  204      0
40d810   0        401efc   0        HDR+S    1000002  204      0
40d3dc   0        401db8   0           S     1000002  204      0
 3 active, 20000 total, 2048 hash buckets

RSAM Version 5.00.UC2  -- On-Line -- Up 10 days 01:43:06 -- 1869 Kbytes

Buffers
address  user     flgs pagenum  memaddr  nslots pgflgs xflgs owner    waitlist
4e97aee3001e0   586300   83     50       400    401efc   0
 11 modified, 200 total, 256 hash buckets, 4096 buffer size

RSAM Version 5.00.UC2  -- On-Line -- Up 10 days 01:50:09 -- 1869 Kbytes

Tblspaces
 n address  flgs ucnt tblnum   physaddr npages nused  npdata nrows  nextns
 0 4e899c   1    1    1000001  100004e  500    114    0      0      1
 1 4e9ae0   1    1    2000001  200004   80     2      0      0      1
 2 4e9be4   1    1    3000001  300004   1020   967    0      0      3
 3 4e9de8   1    1    4000001  400004   310    79     0      0      1
14 4ea7d4   1    1    30002e4  301a96   47     47     30     941    1
232 4b853c   1    1    3000218  300d98   18     17     14     558    2
233 4b8640   1    1    300072c  300d98   4      3      2      152    1
235 4b8948   1    1    3000fc7  300d33   4      2      1      3      1
236 4b894c   1    1    3002263  300e11   8      5      3      148    1
241 4b8e60   1    2    3000200  300d6d   24     20     11     383    3
244 4b916c   1    1    3000301  300d6d   72     70     50     4827   9
245 4b9170   1    1    3000148  300d1e   4      2      1      24     1
248 4b957c   1    1    3000302  301b74   8      2      1      1      1
249 4b9680   1    1    3000203  300d4f   16     15     4      382    2
250 4b9794   1    1    3000140  300d4c   4      2      1      1      1
254 4b9d94   1    1    3000227  300d93   9      7      6      283    1
255 4b9c98   1    1    3000218  300d89   36     33     21     1241   1
 17 active, 500 total, 128 hash buckets

RSAM Version 5.00.UC2  -- On-Line -- Up 10 days 07:40:47 -- 1888 Kbytes

Dbspaces
address  number   flags    fchunk   nchunks  flags    owner    name
40321c   1        1        1        1        N        informix rootdbs
40324d   2        1        2        1        N        informix logdbs
40327c   3        1        3        2        N        informix 2gurgen
4032ad   4        1        4        1        N        informix sm101
 4 active, 8 total

Chunks
address  chk/dbs offset   size     free     bpages   flags pathname
401894   1   1   0        16750    4000              PO-   /dev/rdsk/zd2s2
401934   2   2   12525    4250     1                 PO-   /dev/rdsk/zd2s5
4019cd   3   3   0        16750    37                PO-   /dev/rdsk/zd2s1
401a64   4   4   0        15250    117               PO-   /dev/rdsk/zd1s0
401afc   5   3   0        17000    1693              PO-   /dev/rdsk/zd2s3
 5 active, 8 total

RSAM Version 5.00.UC2  -- On-Line -- Up 10 days 08:31:03 -- 1889 Kbytes

Physical Logging
Buffer bufused  bufsize  numpages numwrits pages/io
  p-1  0        8        3813     549      6.95
     phybegin physize  phypos   phyused  %used
     200034d  500     49c      0        0.00

Logical Logging
Buffer bufused  bufsize  numrecs  numpages numwrits recs/pages pages/io
  l-1  0        8        19331    611      568      31.6        1.1

address  number   flags    uniqid   begin    size     used     %used
4ca57c   3        F------  0        200231   500      0        0.00
4ca598   4        F------  0        233445   500      0        0.00
4ca5b4   7        U---C-L  96       250639   500      73       14.60
4ca5d0   8        F------  0        2008id   500      0        0.00

RSAM Version 5.00.UC2  -- On-Line -- Up 10 days 09:10:04 -- 1888 Kbytes

Profile
dskreads pagreads bufreads %cached dskwrits pagwrits bufwrits %cached
11790    20029    3661718  99.09   4749     9964     425482   98.89

isamtot  open     start    read     write    rewrite  delete   commit   rollbk
3749462  90799    164911   1634873  127250   550      71648    287965   52

ovtbls   ovlock   ovuser   ovbuff   usercpu  syscpu   numckpts flushes
1        0        1        0        5323.20  422.97   134      2208

bufwaits lockwaits lockreqs deadlks  dltouts  lchwaits ckpwaits compress
29       0         4273514  1        0        10       13       4358
```

collects information, and includes explanations as to how that information is manipulated to be put into the structure. The structures holding this information are as follows:

```
struct display_scale {
        char ds_name(MAX);
        XmString ds_label;
        char *ds_style;
        double ds_value;
        int ds_amount;
        char ds_redzone;
        char ds_alarm;
};
```

ds_name is the name of the shared memory component to be displayed; ds_label is that same name in a form printable by Xt Intrinsics; ds_style tells if the monitoring device is a dial or simply an alarm window; ds_value is the true value of the component to be displayed; ds_amount is the adjusted value that is actually displayed on the dial; and ds_redzone and ds_alarm are set to 'y' if a component's value reaches certain points. When "Monitor" was originally written, the intention was to change the color of the dial if a value reached a given point on the dial. However, through experiment it was discovered that the SGI Indigo R3000 does not properly support the color changes necessary to make this feature effective. Therefore, the ds_redzone field is set for every component, but it is never utilized. I could have removed this field from the structure, but since this application can be run on machines other than the SGI's, I left it in to cut down on the work needed to allow for

color changes on other systems.

Once all relevant information has been stored in the array of structures, **dial** starts displaying the dials. The program does this work by first initializing a top level widget. This step sets up a connection to Xt Intrinsics which allows any widgets that are used to be displayed to the screen. Second, a base window is created which will organize the button and dial widgets in rows and columns. Next we create three buttons used to control the "Monitor" program: Pause, Continue, and Stop. These buttons, when activated, make a call to their respective functions: Pause causes the program to hold its current dial positions, Continue starts the dials moving again, and Stop kills the program.

Now that the preliminary work has been done, **dial** moves into its main loop to create a dial for each component of shared memory whose ds_style field is set to "dial". Attached to the top of each dial is a push button which displays the name of the shared memory component monitored by that dial. If the button is activated, the **alarm** process is called and told to display help information about that dial. In any event, the dials and buttons are displayed in rows and columns, and their indicators point to the position set by the ds_amount field in the array of structures. As the program runs, it updates the values of the dials at regular intervals, re-executing the **tbstat** utility and getting new information to be stored in the array of structures. Figure-6 shows an

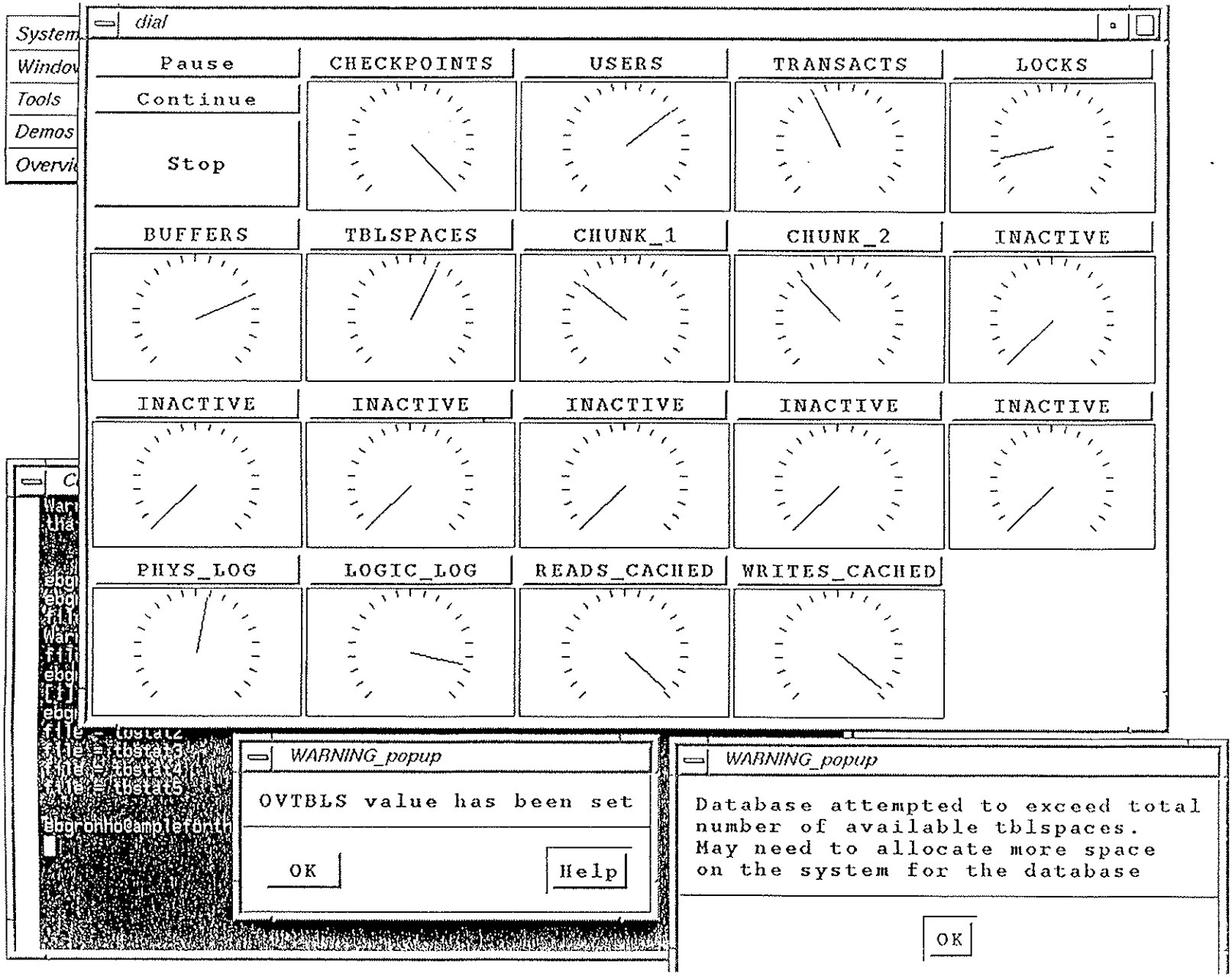


FIGURE--6 The Monitor Program

example of what the "Monitor" program might look like at run time.

In addition to being called by some kind of user activity, **alarm** is also called automatically when the ds_alarm field has been set to 'y'. As **dial** gathers new information, it resets the fields of each structure in the array. When the ds_alarm field is set to 'y', **dial** creates a child, or subprocess, which calls the **alarm** program and requests that an alarm be created and displayed to the screen. This creation of the child process is shown in Figure-7. The important point to understand about creating a child process is that it creates an exact replica of the program currently running, maintaining all values as they have been set by the main program. Once this replica has been made, the program that created the subprocess becomes known as the parent process. **Dial** creates its child process by using the "fork" command, shown below:

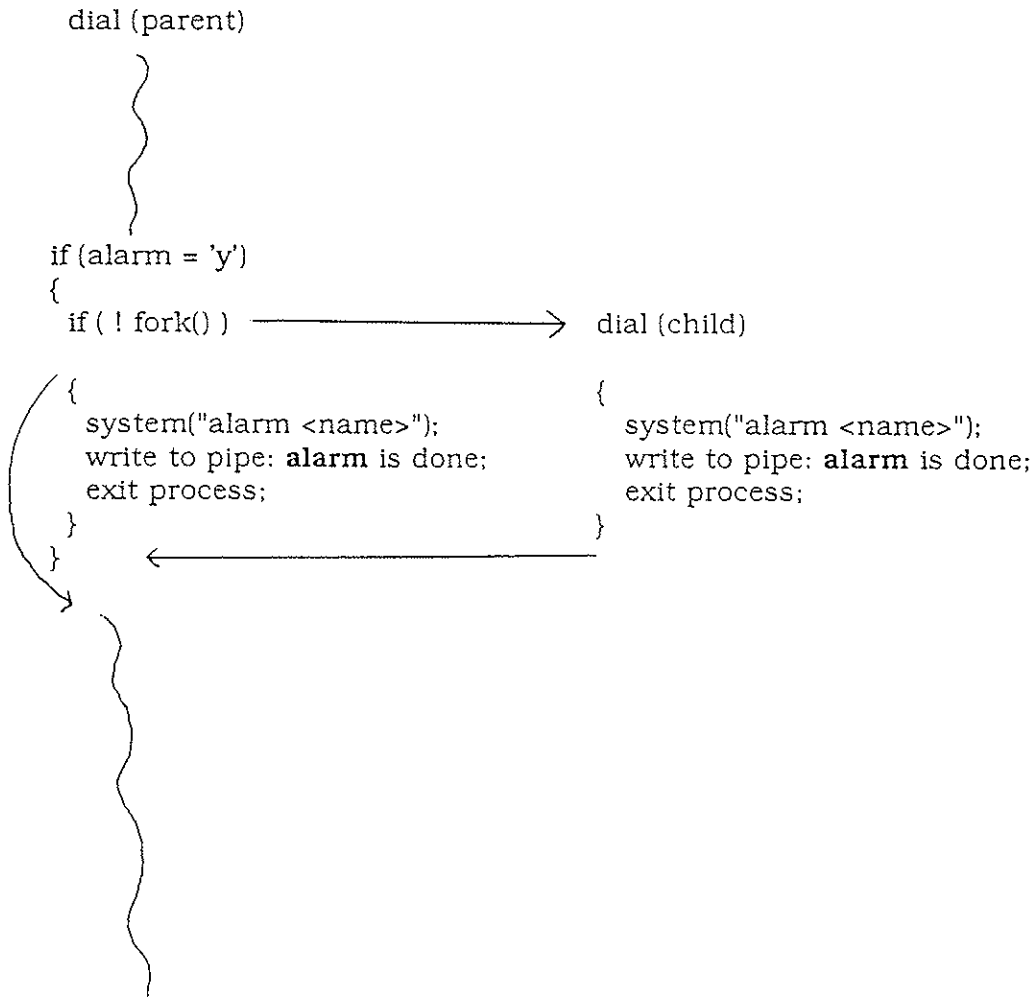
```
if ( ! fork() )
{
<do some kind of work>
}
```

By using this code, the program creates a subprocess and only runs the commands between the braces in the child process. That is, the parent process does not make a call to **alarm**, it simply continues on with the code that follows the last brace and leaves the alarm call to the child.

When the child calls **alarm**, it uses the "system" command just like the call to **tbstat** does. The child sends an argument with **alarm** to notify **alarm** which dial caused the alarm to be created. **Alarm** then

FIGURE 7

Creation of Child Process to call **alarm**



pops up the proper alarm message and waits until the window is killed. Once the window has been killed, control returns to the child process, which then notifies the parent process through a Unix pipe that **alarm** is finished. The parent process needs to be notified about **alarm's** conclusion because "Monitor" is designed not to display an alarm for a dial if an alarm is currently being displayed. This feature reduces the risk of running out of memory because alarms cannot continually pile up on themselves. Once the child process notifies the parent through the pipe, the child ends its own process, leaving the parent process to run alone once again.

The Unix pipe has been mentioned twice: once in the INFORMIX OnLine client/server model, and here with **alarm**. Pipes and child processes make this program run much more efficiently than if the program tried to handle all of the widget creation on its own because they eliminate a large amount of overhead. By having the alarms generated by a separate program, the dials can continue to be updated at regular intervals without having to wait for the alarms to be popped up. Pipes are important because they allow child processes to communicate with their parent processes. Although a child process is running the same program as its parent, the child is a completely independent process. Therefore there must be a link between the two processes if information is to be passed. The pipe is that link. A process

can send information into the pipe and it can come out "the other end" into the parent process. This information is then usable in the process that received it, and in the case of the **dial** parent process, that information means an alarm window may be displayed for the dial whose alarm window was just closed.

Improvements

There are a few improvements "Monitor" could undergo to become an even better and more user-friendly application. Of course as the program is used more often, other improvements will be certain to surface, but the following improvements are the ones which I found to be the most important. First of all, it would be convenient to have the color system work to provide a two levels of warning to the system administrator. Currently, if a dial's indicator reaches a certain position, a warning window is displayed telling the user that if something is not changed soon, there is a chance that the system could crash. Having a dial change color before its indicator reaches a position that triggers an alarm could tell the user to watch that dial's shared memory component more closely. The changing of the color would not mean that something must be done immediately to fix a current problem, it would only be used to raise awareness to that particular area.

Another improvement that might be useful for the system administrator would be to display the actual tbstat information relevant

to a particular dial when the button above that dial is activated. This feature would still allow the user to get a general help message which explains what the dial is monitoring, but it would also tell the user exactly what the status of that dial's shared memory component is. Although the dial is extremely helpful for displaying the general status of its particular component, providing this additional information would give the system administrator the precise data necessary to make informed decisions about the efficiency of the system.

One final improvement could be to create a popup window which allows the user to change the time interval used to update the dials. It may be useful to change this interval as the program is running because there are times when the administrator will want to monitor the system more closely than other times, especially if a dial's color changes or if an alarm window is displayed. When the system seems to be running smoothly, it may not be necessary to update the dials nearly as often. The interval could then be increased to decrease the amount of work being done on the system. Currently the interval can only be controlled at the command line by following the **dial** command with an integer argument for time in seconds. If no argument is given, the program runs at the default of five seconds.

Conclusion

Even though the program could use improvements, "Monitor" is

certainly a useful program for system administrators. Not only does the program display information that monitors the system, but it displays it with simplicity. Dials are easy to read, popup windows can be eliminated when they are not needed, and the three command buttons make the program easy to control. "Monitor" is also a very general program, which means that it can be modified with relatively little work to display any kind of system information, not just information relating to the INFORMIX OnLine Database. Furthermore, since other programs can be written to simulate the type of information that "Monitor" displays, the program is very portable and can be shown on any system that supports C and the X Window System. In short, a graphical application like "Monitor" could be the type of tool that makes evaluating system efficiency easier because components are monitored independently. The system administrator no longer needs to search through cluttered files of information to understand how the system is using its shared memory components. With "Monitor", the information is made available at the click of a button.

BIBLIOGRAPHY

Barkakati, Nabajyoti. *X Window System Programming*, 1st edition. Carmel, Indiana: SAMS, 1991.

INFORMIX OnLine Administrator's Guide, Version 5.0. Menlo Park, CA: Informix Software, Inc., 1991.

INFORMIX OnLine System Administration. Burnsville, MN: Compiled by Systems Migration, Inc., 1992.

Müldner, Tomasz and Peter W. Steele. *C as a Second Language: For Native Speakers of Pascal.* Reading, Massachusetts: Addison-Wesley, 1988.

Open Software Foundation. *OSF/Motif Programmer's Guide.* Englewood Cliffs, New Jersey: PTR Prentice-Hall, Inc., 1991.

Waite, Mitchell and Stephen Prata. *The Waite Group's New C Primer Plus*, 1st edition. Carmel, Indiana: SAMS, 1990.

Young, Douglas A. *The X Window System: Programming and Applications with Xt*, OSF/Motif edition. Englewood Cliffs, New Jersey: PTR Prentice-Hall, Inc., 1990.

Young, Douglas A. *Object-Oriented Programming with C++ and OSF/Motif.* Englewood Cliffs, New Jersey: PTR Prentice-Hall, Inc., 1988.

Appendices

A. Shared Memory Information Monitored by Dials

B. The Monitor Program
   dial.c
   get_display_info.c
   mon_fcts.c
   mon_interface.h
   mon_defs.h
   Makefile
   alarm.c
   Make_alarm
   tbstat.c
   Dial.c
   Dial.h
   DialP.h

Appendix A

SHARED MEMORY INFORMATION MONITORED BY DIALS

CHECKPOINTS: flush out the buffer to disk; system back up. This dial displays (average of all recorded checkpoints/CKPTINTVL) * 100. CKPTINTVL is the minimum checkpoint interval recommended. The dial hits its red zone if this display amount is less than half of CKPTINTVL; it hits an alarm if its amount reaches the red zone for five or move checkpoint times in a row.

USERS: number of processes running on the system. This dial displays (active users/total users allowed) * 100. The dial hits its red zone and triggers and alarm if the number of active users comes within five of the total users allowed.

TRANSACTS: shows number of all transactions associated with users on the system. This dial displays (active transactions/total transactions allowed) * 100. The dial hits its red zone if this amount is greater than 85; it hits an alarm if the amount exceeds 95.

LOCKS: used to manage access to shared memory buffers by limiting the number of users to buffers or by limiting the type of access users have to buffers. This dial displays (active locks/total locks available) * 100. The dial hits its red zone if this amount is greater than 85; it hits an alarm if the amount exceeds 95.

BUFFERS: system memory; Big Buffer = 8 pages Reg. Buffer = 1 page. This dial displays (modified buffers/total buffers available) * 100. The dial hits its red zone if this amount is greater than 85; it hits an alarm if the amount exceeds 95.

TBLSPACES: amount of disk spaces allocated to a specific database table. This dial displays (active tblspaces/total tblspaces available) * 100. The dial hits its red zone if this amount is greater than 85; it hits an alarm if it exceeds 95.

CHUNKS: largest unit of physical disk dedicated to OnLine data storage; measured in pages. All chunks available have their own dial: active chunk dials display ((chunk size - pages free)/chunk size) * 100. Each active chunk dial hits its red zone if this amount is less than 90; an alarm is set if all active chunks have hit their red zones. Inactive chunks are

labeled as such and display an amount of zero.

PHYSICAL LOGS: collections of contiguous pages on disk used to record what pages looked like before they were modified; used for recovery purposes. This dial displays ((pages/io)/bufsize) * 100. The dial hits it red zone and triggers an alarm if this amount falls below 75.

LOGICAL LOGS: collections of contiguous pages on disk used to store all modifications since last system backup. This dial displays ((pages/io)/bufsize) * 100. The dial hits it red zone and triggers an alarm if this amount falls below 75.

READS CACHED: comparison of the number of times data is read from memory (buffers) versus number of reads from disk. This dial displays ((bufreads - dskreads)/bufreads) * 100. The dial hits its red zone and triggers an alarm if this amount falls below 95.

WRITES CACHED: comparison of the number of times data is written to memory (buffers) versus number of writes to disk. This dial displays ((bufwrits - dskwrits)/bufwrits) * 100. The dial hits its red zone and triggers an alarm if this amount falls below 85.

The remaining five components are not represented by dials. They all trigger an alarm if their values exceed zero.

OVTBLS: number of times OnLine attempted to exceed maximum number of available tblspaces.

OVLOCK: number of times OnLine attempted to exceed maximum number of locks.

OVUSER: number of times a user attempted to exceed maximum number of users.

OVBUFF: number of times OnLine attempted to exceed maximum number of shared memory buffers.

DEADLKS: increments each time a potential deadlock is detected and prevented.

Appendix B

The Monitor Program--Code

dial.c
get_display_info.c
mon_fcts.c
mon_interface.h
mon_defs.h
Makefile
alarm.c
Make_alarm
tbstat.c
Dial.c
Dial.h
DialP.h

```
/*****************************************************************************
 * dial.c : test the Dial widget class
 *         From:
 *                  The X Window System,
 *              Programming and Applications with Xt
 *                  OSF/Motif Edition
 *         by
 *                Douglas Young
 *              Prentice Hall, 1990
 *
 *                  Example described on pages: 362-364
 *
 *
 *  Copyright 1989 by Prentice Hall
 *  All Rights Reserved
 *
 * This code is based on the OSF/Motif widget set and the X Window System
 *
 * Permission to use, copy, modify, and distribute this software for
 * any purpose and without fee is hereby granted, provided that the above
 * copyright notice appear in all copies and that both the copyright notice
 * and this permission notice appear in supporting documentation.
 *
 * Prentice Hall and the author disclaim all warranties with regard to
 * this software, including all implied warranties of merchantability and fitne
 * In no event shall Prentice Hall or the author be liable for any special,
 * indirect or cosequential damages or any damages whatsoever resulting from
 * loss of use, data or profits, whether in an action of contract, negligence
 * or other tortious action, arising out of or in connection with the use
 * or performance of this software.
 *
 * Open Software Foundation is a trademark of The Open Software Foundation, Inc
 * OSF is a trademark of Open Software Foundation, Inc.
 * OSF/Motif is a trademark of Open Software Foundation, Inc.
 * Motif is a trademark of Open Software Foundation, Inc.
 * DEC is a registered trademark of Digital Equipment Corporation
 * HP is a registered trademark of the Hewlett Packard Company
 * DIGITAL is a registered trademark of Digital Equipment Corporation
 * X Window System is a trademark of the Massachusetts Institute of Technology
 *
 *
 *     Modified by:  Eric B. Gronholz
 *                   Systems Migration, Inc.
 *                   501 E. Hwy 13 Suite 112
 *                   Burnsville, MN  55337
 *
 *     Date Modified: April/May, 1993
 *****************************************************************************/

#include <unistd.h>
#include <fcntl.h>
#include <Xm/Label.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include "Dial.h"
```



```
#include "mon_interface.h"

void call_tbstat();
void update_dials();
void pause_callback();
void proceed_callback();
void stop_callback();
void dial_info_callback();
extern void send_message();

int set_intvl;
int set_interval = 5000;
int alarm_pipe[MAX_SCALES][2];
int alarm_time[MAX_SCALES];     /* used to keep track of alarms */
char command[100];              /* for call to create alarm window */
char is_done,                   /* used for pipe read */
     done = 'y';                /* used for pipe write */
int stop = 0;                   /* flag for pause/proceed buttons */
int m, n, c, r;                 /* geometry variables used in alarm call */
char info_msg[MAX_LINES][MAX];
struct help_info {
     char hi_name[MAX];
     long hi_pos;
};

int main(argc, argv)
  int   argc;
  char *argv[];
{

  Widget toplevel,
         rowcol,
         pause,
         proceed,
         stop_widg,
         button_form,
         form[MAX_SCALES];
  Arg    pause_args[MAX_ARGS],
         proceed_args[MAX_ARGS],
         stop_args[MAX_ARGS],
         dial_args[MAX_ARGS],
         label_args[MAX_ARGS];
 struct help_info hlp_info;
 struct display_scale dials[MAX_SCALES];
  long dial_index;
  int d, lab, n, i, q;


  /*********************************************************/
  /*  Have user specify time out interval at command line.  */
  /*  User enters time in seconds, program converts time to */
  /*  milliseconds.  Default is five seconds (5000 ms).     */
  /*********************************************************/
  if (argc > 1)
  {
     set_intvl = atoi(argv[argc-1]);
```

```
    set_interval = set_intvl * 1000;
}

/************************************************/
/* Initialize pipes used and sets the O_NONBLOCK */
/* flag to allow program to continue without     */
/* waiting for output from pipe.                 */
/************************************************/
 for (i = 0; i < MAX_SCALES; i++)
 {
    if (pipe(alarm_pipe[i]) != 0)
    {
       printf("Pipe was not created.  Kill process\n");
       exit(1);
    }
    else
       fcntl(alarm_pipe[i][0], F_SETFL, FNONBLK);
}

/*****************************/
/*** Get info from tbstat ***/
/*****************************/
call_tbstat(dials, &dial_index);
upd_info.d_index = dial_index;

/*********************************/
/*** Initialize the Intrinsics ***/
/*********************************/
toplevel = XtInitialize(argv[0], "Dial", NULL, 0, &argc, argv);

/*********************************/
/*** Set arguments for basewindow ***/
/*********************************/
rowcol = XtCreateManagedWidget("rowcol", xmRowColumnWidgetClass,
                               toplevel, NULL, 0);

/*********************************/
/*** Make form widget for buttons ***/
/*********************************/
button_form = XtCreateManagedWidget("but_form", xmFormWidgetClass,
                                    rowcol, NULL, 0);

/*************************************************/
/*** Create pause/proceed/stop_widg pushbuttons:  ***/
/*** pause sets stop to 1, proceed sets stop to 0,***/
/*** stop_widg exits program.                     ***/
/*************************************************/
pause = XtCreateManagedWidget("Pause", xmPushButtonWidgetClass,
                              button_form, NULL, 0);
XtAddCallback(pause, XmNactivateCallback, pause_callback, NULL);

proceed = XtCreateManagedWidget("Continue", xmPushButtonWidgetClass,
                                button_form, NULL, 0);
XtAddCallback(proceed, XmNactivateCallback, proceed_callback, NULL);

stop_widg = XtCreateManagedWidget("Stop", xmPushButtonWidgetClass,
```



```
                                  button_form, NULL, 0);
XtAddCallback(stop_widg, XmNactivateCallback, stop_callback, NULL);

/*********************************/
/*** Format buttons into one form ***/
/*********************************/
n = 0;
XtSetArg(pause_args[n], XmNtopAttachment, XmATTACH_FORM); n++;
XtSetArg(pause_args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(pause_args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetValues(pause, pause_args, n);

n = 0;
XtSetArg(proceed_args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(proceed_args[n], XmNtopWidget, pause); n++;
XtSetArg(proceed_args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(proceed_args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetValues(proceed, proceed_args, n);

n = 0;
XtSetArg(stop_args[n], XmNtopAttachment, XmATTACH_WIDGET); n++;
XtSetArg(stop_args[n], XmNtopWidget, proceed); n++;
XtSetArg(stop_args[n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg(stop_args[n], XmNrightAttachment, XmATTACH_FORM); n++;
XtSetArg(stop_args[n], XmNbottomAttachment, XmATTACH_FORM); n++;
XtSetValues(stop_widg, stop_args, n);


/*********************************************************/
/*** Create form widget to combine dials and buttons    ***/
/*** Create dial and button widgets, assigning initial  ***/
/***  values from dials struct array                    ***/
/*********************************************************/
for (i=0; i < dial_index; i++)
{
      if (strcmp(dials[i].ds_style, "dial") == 0)
      {
         form[i] = XtCreateManagedWidget(dials[i].ds_name,
                                         xmFormWidgetClass,
                                         rowcol, NULL, 0);

         /*******************/
         /*** Create dials ***/
         /*******************/
         XtSetArg(dial_args[0], XtNposition, dials[i].ds_amount);
         upd_info.dial_widg[i] = XtCreateManagedWidget(dials[i].ds_name,
                                                       XsdialWidgetClass,
                                                       form[i], NULL, 0);

         /*********************************/
         /*** Create buttons above dials ***/
         /*********************************/
         XtSetArg(label_args[0], XmNlabelString, dials[i].ds_label);
         upd_info.label_widg[i] = XtCreateManagedWidget(dials[i].ds_name,
                                                        xmPushButtonWidgetClass,
                                                        form[i], NULL, 0);
```

```
            XtAddCallback(upd_info.label_widg[i], XmNactivateCallback,
                          dial_info_callback, dials[i].ds_name);


            /*******************************************************/
            /* Set resources so label widget is attached to top of  */
            /* corresponding dial widget.  Also change colors as    */
            /* needed for redzones and inactive chunks              */
            /*******************************************************/
            d = 1;
            lab = 1;
            XtSetArg(label_args[lab], XmNtopAttachment, XmATTACH_FORM); lab++;
            XtSetArg(label_args[lab], XmNleftAttachment, XmATTACH_FORM); lab++;
            XtSetArg(label_args[lab], XmNrightAttachment, XmATTACH_FORM); lab++;
            XtSetArg(dial_args[d], XmNtopAttachment, XmATTACH_WIDGET); d++;
            XtSetArg(dial_args[d], XmNtopWidget, upd_info.label_widg[i]); d++;
            XtSetArg(dial_args[d], XmNleftAttachment, XmATTACH_FORM); d++;
            XtSetArg(dial_args[d], XmNrightAttachment, XmATTACH_FORM); d++;
            XtSetArg(dial_args[d], XmNbottomAttachment, XmATTACH_FORM); d++;

            XtSetValues(upd_info.label_widg[i], label_args, lab);
            XtSetValues(upd_info.dial_widg[i], dial_args, d);

         }  /* end if style == dial */

         if (dials[i].ds_alarm == 'y')
         {
            /*****************************************************************/
            /* Create a popup alarm window using a fork process for all dials */
            /* which set an alarm.  Popup window created by a system  call    */
            /* and pipe is written to when alarm window has been killed.      */
            /*****************************************************************/
            if (alarm_time[i] == 0)
            {
               /*************************************************/
               /* Increment alarm_time[i] so process knows that    */
               /* an alarm window exists for this member of dials  */
               /* dials array.  Launch alarm window and then write */
               /* to pipe, notifying other process that alarm      */
               /* window has been killed.                          */
               /*************************************************/
               alarm_time[i]++;
               if ( !fork() )            /* only do work in forked process */
               {
                  /* set upper left hand corner */
                  if ( ((i + 1) % COLUMNS) > (COLUMNS - 1) )
                     c = i + 1;
                  else
                     c = COLUMNS - ((i + 1) % COLUMNS);

                  r = 0;
                  if ( (i % ROWS) == 0 )
                    r = r + HEIGHT;

                  /* set size of alarm window */
```



```
                  m = (int) WIDTH;
                  n = (int) HEIGHT;

                  sprintf(command, "alarm -geom=%ld+%ld+%ld+%ld %s",
                        m, n, c, r, dials[i].ds_name);
                  system (command);
                  write(alarm_pipe[i][1], &done, sizeof(char));
                  exit(0);
               }
            }
         }
      } /*** end for loop ***/

   XtAddTimeOut(set_interval, update_dials, &upd_info);

   XtRealizeWidget(toplevel);
   XtMainLoop();

}  /*** end main ***/


/*****************************************************************************/
/*****************************************************************************/
void pause_callback(w, client_data, call_data)
  Widget                w;
  caddr_t               client_data;
  XmAnyCallbackStruct   *call_data;

{
       stop = 1;
}

/*****************************************************************************/
void proceed_callback(w, client_data, call_data)
  Widget                w;
  caddr_t               client_data;
  XmAnyCallbackStruct   *call_data;

{
       stop = 0;
}

/*****************************************************************************/
void stop_callback(w, client_data, call_data)
  Widget                w;
  caddr_t               client_data;
  XmAnyCallbackStruct   *call_data;

{
       exit(1);
}

/*****************************************************************************/
void dial_info_callback(w, dialname, call_data)
  Widget                w;
  char                  *dialname;
```

```
  XmAnyCallbackStruct   *call_data;

{  /** print out the help window for a specific dial **/

        if ( !fork() )          /* only do work in forked process */
        {
           sprintf(command, "alarm help %s", dialname);
           system(command);
           exit(0);

     } /** end if !fork() **/
}

/*************************************************************************/
void call_tbstat(dials, dial_index)
   struct display_scale *dials;
   long *dial_index;

{
   FILE *fp;
   long index;

   /*** make call to tbstat from command line and send output to a file ***/
   system("tbstat > tbstat.info");

   if ((fp = fopen("tbstat.info", "r")) == NULL) {
      printf("cannot open \"tbstat.info\"\n");
      exit(1);
   };


   /*** place data from tbstat into dials struct array ***/
   get_display_info(fp, dials, dial_index);


} /*** end call_tbstat ***/

/*************************************************************************/
 void update_dials(upd_info, id)
     struct update_info *upd_info;
     XtIntervalId id;
{
   Arg dial_arg[MAX_ARGS],
       label_arg[MAX_ARGS];
   int i, lab, d;


if (stop == 0)
{
   call_tbstat(upd_info->dials, &upd_info->d_index);

   /* change labels, dial positions, dial backgrounds */
   for (i = 0; i < upd_info->d_index; i++)
   {
      if (strcmp(upd_info->dials[i].ds_style, "dial") == 0)
      {
```



```
        /***********************************/
        /* Update labels and dial positions */
        /***********************************/
        d = 0;
        lab = 0;
        XtSetArg(label_arg[lab], XmNlabelString,
                upd_info->dials[i].ds_label);lab++;
        XtSetArg(dial_arg[d], XtNposition,
                upd_info->dials[i].ds_amount);d++;

        XtSetValues(upd_info->label_widg[i], label_arg, lab);
        XtSetValues(upd_info->dial_widg[i], dial_arg, d);


        if (upd_info->dials[i].ds_alarm == 'y')
        {
          /*****************************************************************
          /* If an alarm window has not been popped up for this alarm,  */
          /* start a new alarm window process.                          */
          /*****************************************************************/
          if (read(alarm_pipe[i][0], &is_done, sizeof(is_done)) != -1)
          {
             alarm_time[i] = 0;
          }

          if (alarm_time[i] == 0)
          {
             /************************************************/
             /* Increment alarm_time[i] so process knows that    */
             /* an alarm window exists for this member of dials  */
             /* dials array.  Launch alarm window and then write */
             /* to pipe, notifying other process that alarm      */
             /* window has been killed.                          */
             /************************************************/
             alarm_time[i] = 1;
             if ( !fork() )
             {
                /* set upper left hand corner */
                if ( ((i + 1) % COLUMNS) > (COLUMNS - 1) )
                   c = i + 1;
                else
                   c = COLUMNS - ((i + 1) % COLUMNS);

                r = 0;
                if ( (i % ROWS) == 0 )
                  r = r + HEIGHT;

                /* set size of alarm window */
                m = (int) WIDTH;
                n = (int) HEIGHT;

                sprintf(command, "alarm -geom=%ld+%ld+%ld+%ld  %s",
                       m, n, c, r, upd_info->dials[i].ds_name);
                system (command);
                write(alarm_pipe[i][1], &done, sizeof(char));
                exit(0);
```

```
            }
         }
         else if (read(alarm_pipe[i][0], &is_done, sizeof(is_done)) != -1)
         {
            alarm_time[i] = 0;
         }

      } /* end if alarm == 'y' */


   } /* end if style == "dial" */
  } /* end for loop */
} /* end if stop statement */

  XtAddTimeOut(set_interval, update_dials, upd_info);


} /* end update_dials */
```

```
/*
 ###############################################################################
 #                              Eric B. Gronholz                               #
 #                              April/May, 1993                                #
 #                                                                             #
 # get_display_info.c -- Gets info from a tbstat file and computes and         #
 #                       sets values in the scales struct for appropriate      #
 #                       item.  Also keeps track of the actual number of       #
 #                       items in the scales struct array.                     #
 #                                                                             #
 # ASSUMPTION:  This procedure assumes that the input file is formatted as     #
 #              the output would be in the tbstat -a command.  Using files     #
 #              formatted in other ways could cause the function to crash.     #
 #                                                                             #
 ###############################################################################
*/

#include "mon_interface.h"    /* external declarations of functions */

void  get_display_info(fp, scales, scale_index)
        FILE *fp;
        struct display_scale scales[];
        long *scale_index;
 {

/*============*/
/* VARIABLES */
/*============*/
        char config_str[MAX];           /* get a line from input file */
        char chunk_var[MAX_CHUNKS];     /* holds string for naming of chunks */
        char chunk_num[MAX_CHUNKS];
        char msg_setup[50];             /* used to help with messages */
        char set_alarm;
        double  ckpt_array[MAX_CKPTS];  /* holds times of all checkpoints */
        double  min_ckpt_intvl,         /* value of CKPTINTVL in tbstat file */
                avg_intvl,              /* avg. of all ckpt. intervals */
                total,                  /* used in get_act_tot fct. call */
                active,                 /* used in get_act_tot fct. call */
                chunk_percent,          /* used in get_chnk_info fct. call */
                chunk_tot_percent,      /* used in get_chnk_info fct. call */
                p_log_percent,          /* used in get_loggings fct. call */
                l_log_percent,          /* used in get_loggings fct. call */
                rd_cache,             /* used in get_percent_cached fct. call */
                wt_cache,             /* used in get_percent_cached fct. call */
                i, n = 0;
        long    actual_max,             /* actual size of scales struct array */
                which_chunk = 1,        /* holds current chunk being processed *
                index = 0,              /* index for scales array */
                ca_index = 0;           /* index for ckpt_array */
        int     below_ckpts,          /* counts times avg ckpt intvl below min */
                which_log = 0,          /* used in get_loggings call */
                ovtbls,                 /* used in get_over_vals fct. call */
                ovlock,                 /* used in get_over_vals fct. call */
                ovuser,                 /* used in get_over_vals fct. call */
                ovbuff,                 /* used in get_over_vals fct. call */
                deadlk;                 /* used in get_deadlk fct. call */
```



```
        char    not_found;
        char    cont;
        XmString dial_label;

/*===============*/
/* BEGIN PROGRAM */
/*===============*/

        /*** Search through tbstat info file for value of CKPTINTVL.
             Move cursor to top of file to allow program to read from
             beginning. ***/

        /*** GET CKPTINTVL VALUE ***/
        min_ckpt_intvl = ckpt_intvl_min(fp);

        fseek(fp, 0L, SEEK_SET);

        /* go through argument file and get lines until line is found that
           has value of a tbstat_var.  When a tbstat item is found, get desired
           data and place that value into the appropriate scales struct array
           position.  Increment actual_max every time a proper tbstat item is
           found. */

        cont = 1;       /* continue flag */
        while (cont && fgets(config_str, MAX, fp) != NULL)
        {
           /*********************************************************************/
           /*** Find and compute avg. CHECKPOINT interval.  Value is put   ***/
           /*** into the scales struct array.                              ***/
           /*********************************************************************/
           if ( strstr(config_str,"Checkpoint Completed") != NULL )
           {
                get_ckpts(config_str, ckpt_array, ca_index);
                ca_index++;

                /* while loop locates position of checkpoints completed */
                not_found = 1;
                while (not_found && fgets(config_str, MAX, fp) != NULL)
                {
                   if (strstr(config_str, "Checkpoint Completed") != NULL)
                   {
                        get_ckpts(config_str, ckpt_array, ca_index);
                        ca_index++;
                   }
                   else
                        not_found = 0;

                } /* end while not_found && fgets */

                /** decrement ca_index because incremented once too often **/
                ca_index = ca_index - 1;

                /** Determine ckpt intervals and number of times below **/
                /**  minimum, setting alarm if necessary               **/
                compute_ckpt_intvl(ckpt_array, ca_index, min_ckpt_intvl,
                                   &avg_intvl, &set_alarm);
```

```
            strcpy(scales[index].ds_name, "CHECKPOINTS");

            /** labels must be XmStrings so they can be displayed **/
            dial_label = XmStringCreate(scales[index].ds_name,
                                        XmSTRING_DEFAULT_CHARSET);
            scales[index].ds_label = dial_label;
            scales[index].ds_style = STYLE_CKPT;
            scales[index].ds_value = avg_intvl;
            if (avg_intvl/min_ckpt_intvl >= 1)
               scales[index].ds_amount = 100;
            else
               scales[index].ds_amount =
                                  (int)((avg_intvl/min_ckpt_intvl)*100);
            if (scales[index].ds_amount <= RED_CKPT)
               scales[index].ds_redzone = 'y';
            else
               scales[index].ds_redzone = 'n';
            if (set_alarm == 'y')
               scales[index].ds_alarm = 'y';

            index++;
            actual_max = index;

         }  /* end if "Message Log File"--checkpoints */


         /*****************************************************************/
         /*** Find and compute USERS values.  Place into scales array. ***/
         /*****************************************************************/
         else if ( strstr(config_str,"Users") != NULL )
         {

            /* while loop locates position of active and total values */
            not_found = 1;
            while (not_found && fgets(config_str, MAX, fp) != NULL)
            {
               get_act_tot(config_str, &active, &total, &not_found);

            } /* end while not_found and fgets != null */

            strcpy(scales[index].ds_name, "USERS");
            dial_label = XmStringCreate(scales[index].ds_name,
                                        XmSTRING_DEFAULT_CHARSET);
            scales[index].ds_label = dial_label;
            scales[index].ds_style = STYLE_USER;
            scales[index].ds_value = active;
            scales[index].ds_amount = (int) ((active/total) * 100);
            if (active >= (total - USERS_REMAIN))
            {
               scales[index].ds_redzone = 'y';
               scales[index].ds_alarm = 'y';
            }
            else
            {
               scales[index].ds_redzone = 'n';
               scales[index].ds_alarm = 'n';
```



```
            }
            index++;
            actual_max = index;

         }  /* end else if for getting users */

         /*****************************************************************/
         /*** Find and compute TRANSACTIONS values.  Put in scales array. ***/
         /*****************************************************************/
         else if ( strstr(config_str,"Transactions") != NULL )
         {

            /* while loop locates position of active and total values */
            not_found = 1;
            while (not_found && fgets(config_str, MAX, fp) != NULL)
            {
               get_act_tot(config_str, &active, &total, &not_found);

            } /* end while not_found and fgets != null */

            strcpy(scales[index].ds_name, "TRANSACTS");
            dial_label = XmStringCreate(scales[index].ds_name,
                                        XmSTRING_DEFAULT_CHARSET);
            scales[index].ds_label = dial_label;
            scales[index].ds_style = STYLE_TRANS;
            scales[index].ds_value = active;
            scales[index].ds_amount = (int) ((active/total) * 100);
            if (scales[index].ds_amount >= RED_TRANS)
               scales[index].ds_redzone = 'y';
            else
               scales[index].ds_redzone = 'n';
            if (scales[index].ds_amount >= ALARM_TRANS)
               scales[index].ds_alarm = 'y';
            else
               scales[index].ds_alarm = 'n';

            index++;
            actual_max = index;

         }  /* end else if for getting transactions */

         /*****************************************************************/
         /*** Find and compute LOCKS values.  Place into scales array. ***/
         /*****************************************************************/
         else if ( strstr(config_str,"Locks") != NULL )
         {
            /* while loop locates position of active and total values */
            not_found = 1;
            while (not_found && fgets(config_str, MAX, fp) != NULL)
            {
               get_act_tot(config_str, &active, &total, &not_found);

            } /* end while not_found and fgets != null */

            strcpy(scales[index].ds_name, "LOCKS");
            dial_label = XmStringCreate(scales[index].ds_name,
```

```
                                        XmSTRING_DEFAULT_CHARSET);
            scales[index].ds_label = dial_label;
            scales[index].ds_style = STYLE_LOCKS;
            scales[index].ds_value = active;
            scales[index].ds_amount = (int) ((active/total) * 100);
            if (scales[index].ds_amount >= RED_LOCKS)
               scales[index].ds_redzone = 'y';
            else
               scales[index].ds_redzone = 'n';
            if (scales[index].ds_amount >= ALARM_LOCKS)
               scales[index].ds_alarm = 'y';
            else
               scales[index].ds_alarm = 'n';

            index++;
            actual_max = index;

         }  /* end else if for getting locks */

         /*****************************************************************/
         /*** Find and compute BUFFERS values.  Place into scales array. ***/
         /*****************************************************************/
         else if ( strstr(config_str,"Buffers") != NULL )
         {
              /* while loop locates position of active and total values */
              not_found = 1;
              while (not_found && fgets(config_str, MAX, fp) != NULL)
              {
                 get_act_tot(config_str, &active, &total, &not_found);

              } /* end while not_found and fgets != null */

              strcpy(scales[index].ds_name, "BUFFERS");
              dial_label = XmStringCreate(scales[index].ds_name,
                                        XmSTRING_DEFAULT_CHARSET);
              scales[index].ds_label = dial_label;
              scales[index].ds_style = STYLE_BUFFS;
              scales[index].ds_value = active;
              scales[index].ds_amount = (int) ((active/total) * 100);
              if (scales[index].ds_amount >= RED_BUFFS)
                 scales[index].ds_redzone = 'y';
              else
                 scales[index].ds_redzone = 'n';
              if (scales[index].ds_amount >= ALARM_BUFFS)
                 scales[index].ds_alarm = 'y';
              else
                 scales[index].ds_alarm = 'n';

              index++;
              actual_max = index;

         }  /* end else if for getting buffers */

         /*****************************************************************/
         /*** Find and compute TBLSPACES values.  Put into scales array. ***/
         /*****************************************************************/
```



```
         else if ( strstr(config_str,"Tblspaces") != NULL )
         {
              /* while loop locates position of active and total values */
              not_found = 1;
              while (not_found && fgets(config_str, MAX, fp) != NULL)
              {
                 get_act_tot(config_str, &active, &total, &not_found);

              } /* end while not_found and fgets != null */

              strcpy(scales[index].ds_name, "TBLSPACES");
              dial_label = XmStringCreate(scales[index].ds_name,
                                        XmSTRING_DEFAULT_CHARSET);
              scales[index].ds_label = dial_label;
              scales[index].ds_style = STYLE_TBLS;
              scales[index].ds_value = active;
              scales[index].ds_amount = (int) ((active/total) * 100);
              if (scales[index].ds_amount >= RED_TBLS)
                 scales[index].ds_redzone = 'y';
              else
                 scales[index].ds_redzone = 'n';
              if (scales[index].ds_amount >= ALARM_TBLS)
                 scales[index].ds_alarm = 'y';
              else
                 scales[index].ds_alarm = 'n';

              index++;
              actual_max = index;

         }  /* end else if for getting tblspaces */

         /*****************************************************************/
         /*** Find and compute CHUNKS values.  Place into scales array. ***/
         /*****************************************************************/

         else if ( strstr(config_str,"Chunks") != NULL )
         {

              /* while loop locates size and free of all active chunks */
              not_found = 1;
              while (not_found && fgets(config_str, MAX, fp) != NULL)
              {

                 /* first step is to read past column headings.  If config_str
                    contains the word "address," get the next config_str.
                    Next, check to see if config_str contains "active".  If
                    it does, all active chunks have been read, so assign space
                    in scales array for any available chunks that are not
                    currently active (i.e. assign space for (total - active)
                    more chunks)  In other words, we want as many chunks in
                    the scales array as there are total chunks available.
                 */
                 if (strstr(config_str,"address") != NULL)
                      continue;
                 else if (strstr(config_str, "active") != NULL)
                 {
```

```
                    get_act_tot(config_str, &active, &total, &not_found);
                    if (total > active)
                    {
                        for (i = active + 1; i <= total; i++)
                        {
                           strcpy(scales[index].ds_name, "INACTIVE");
                           dial_label = XmStringCreate(scales[index].ds_name,
                                                       mSTRING_DEFAULT_CHARSET);
                           scales[index].ds_label = dial_label;
                           scales[index].ds_style = "dial";
                           scales[index].ds_value = 0.0;
                           scales[index].ds_amount = 0;
                           scales[index].ds_redzone = 'n';
                           scales[index].ds_alarm = 'n';
                           index++;
                        } /* end for loop */
                    } /* end if statement */
                } /* end else if statement */
                else
                {
                     get_chnk_info(config_str, &chunk_percent,
                                   &chunk_tot_percent);

                     /* make variable name for the correct chunk, i.e. if
                        processing first chunk, ds_name = "CHUNK_1"; second,
                        ds_name = "CHUNK_2", and so on. */

                     sprintf(chunk_var, "CHUNK_");
                     sprintf(chunk_num, "%d", which_chunk);
                     strcat(chunk_var, chunk_num);
                     strcpy(scales[index].ds_name, chunk_var);
                     dial_label = XmStringCreate(scales[index].ds_name,
                                                 XmSTRING_DEFAULT_CHARSET);
                     scales[index].ds_label = dial_label;
                     scales[index].ds_style = STYLE_CHUNKS;
                     scales[index].ds_value = chunk_percent;
                     scales[index].ds_amount = (int) (chunk_percent * 100);
                     if (scales[index].ds_amount >= RED_CHUNKS)
                        scales[index].ds_redzone = 'y';
                     else
                        scales[index].ds_redzone = 'n';
                     scales[index].ds_alarm = 'n';

                     which_chunk++;
                     index++;
                     actual_max = index;

                } /* end else */

            } /* end while not_found and fgets != null */

            /* Alarm for chunks only set in final position of array where
               chunks are stored.  Since index has been incremented in the
               while loop, must get back to last position accessed in
               array.
            */
```



```
                if ( (int) (chunk_tot_percent * 100) >= ALARM_CHUNKS)
                     scales[index - 1].ds_alarm = 'y';

            }  /* end else if for getting chunks */

            /*****************************************************************/
            /*** Find and compute LOGGING values.  Place into scales array. ***/
            /*****************************************************************/
            else if ( strstr(config_str,"Physical Logging") != NULL ||
                      strstr(config_str, "Logical Loggin") != NULL)
            {
                 if (strstr(config_str, "Physical Logging") != NULL)
                 {
                    which_log = 0;       /* tell fct p. log is being processed */
                 }
                 else if (strstr(config_str, "Logical Logging") != NULL)
                 {
                    which_log = 1;       /* tell fct l. log is being processed */
                 }

                 /* while loop locates bufsize and pages_io of loggings */
                 not_found = 1;
                 while (not_found && fgets(config_str, MAX, fp) != NULL)
                 {

                    /* first step is to read past column headings. If config_str
                       contains the word "Buffer," get the next config_str. */

                    if (strstr(config_str,"Buffer") != NULL)
                       continue;
                    else
                    {
                         get_loggings(config_str, which_log, &p_log_percent,
                                      &l_log_percent, &not_found);

                         if (which_log == 0)
                         {
                            strcpy(scales[index].ds_name, "PHYS_LOG");
                            dial_label = XmStringCreate(scales[index].ds_name,
                                                        XmSTRING_DEFAULT_CHARSET);
                            scales[index].ds_label = dial_label;
                            scales[index].ds_style = STYLE_LOG;
                            scales[index].ds_value = p_log_percent;
                            scales[index].ds_amount = (int)(p_log_percent * 100);
                            if (scales[index].ds_amount <= ALARM_LOG)
                            {
                               scales[index].ds_redzone = 'y';
                               scales[index].ds_alarm = 'y';
                            }
                            else
                            {
                               scales[index].ds_redzone ='n';
                               scales[index].ds_alarm = 'n';
                            }
                         }
```

```
                    else
                    {
                       strcpy(scales[index].ds_name, "LOGIC_LOG");
                       dial_label = XmStringCreate(scales[index].ds_name,
                                                   XmSTRING_DEFAULT_CHARSET);
                       scales[index].ds_label = dial_label;
                       scales[index].ds_style = STYLE_LOG;
                       scales[index].ds_value = l_log_percent;
                       scales[index].ds_amount = (int)(l_log_percent * 100);
                       if (scales[index].ds_amount <= ALARM_LOG)
                       {
                          scales[index].ds_redzone = 'y';
                          scales[index].ds_alarm = 'y';
                       }
                       else
                       {
                          scales[index].ds_redzone ='n';
                          scales[index].ds_alarm = 'n';
                       }
                    }

                    index++;
                    actual_max = index;

                } /* end else */

            } /* end while not_found and fgets != null */

        } /* end else if for getting logging */

        /*********************************************************************/
        /*** Find both types of %cached in profile.  Put in scales array.***/
        /*********************************************************************/
        else if ( strstr(config_str,"Profile") != NULL)
        {
            /* while loop locates bufreads' %cached and bufwrits' %cached */
            not_found = 1;
            while (not_found && fgets(config_str, MAX, fp) != NULL)
            {
               /* first step is to read past column headings. If config_str
                  contains the word "dskreads," get the next config_str. */

               if (strstr(config_str,"dskreads") != NULL)
               {
                    continue;
               }
               else
               {
                    get_percent_cached(config_str, &rd_cache, &wt_cache,
                                       &not_found);
               } /* end else */

            } /* end while not_found and fgets != null */

            /* store bufreads %cached in scales array */
            strcpy(scales[index].ds_name, "READS_CACHED");
```



```
                    dial_label = XmStringCreate(scales[index].ds_name,
                                                XmSTRING_DEFAULT_CHARSET);
                    scales[index].ds_label = dial_label;
                    scales[index].ds_style = STYLE_CACHE;
                    scales[index].ds_value = rd_cache;
                    scales[index].ds_amount = (int) (rd_cache);
                    if (scales[index].ds_amount < ALARM_RD_CACHE)
                    {
                       scales[index].ds_redzone = 'y';
                       scales[index].ds_alarm = 'y';
                    }
                    else
                    {
                       scales[index].ds_redzone = 'n';
                       scales[index].ds_alarm = 'n';
                    }
                    index++;
                    actual_max = index;

                    /* store bufwrits %cached in scales array */
                    strcpy(scales[index].ds_name, "WRITES_CACHED");
                    dial_label = XmStringCreate(scales[index].ds_name,
                                                XmSTRING_DEFAULT_CHARSET);
                    scales[index].ds_label = dial_label;
                    scales[index].ds_style = STYLE_CACHE;
                    scales[index].ds_value = wt_cache;
                    scales[index].ds_amount = (int) (wt_cache);
                    if (scales[index].ds_amount < ALARM_WT_CACHE)
                    {
                       scales[index].ds_redzone = 'y';
                       scales[index].ds_alarm = 'y';
                    }
                    else
                    {
                       scales[index].ds_redzone = 'n';
                       scales[index].ds_alarm = 'n';
                    }
                    index++;
                    actual_max = index;


            } /* end else if for getting percent cached */


            /*********************************************************************/
            /*** Find and report OVER values.  Place into scales array.  ***/
            /*********************************************************************/
            else if ( strstr(config_str,"ovtbls") != NULL)
            {
                /* while loop locates bufsize and pages_io of loggings */
                not_found = 1;
                while (not_found && fgets(config_str, MAX, fp) != NULL)
                {

                   get_over_vals(config_str, &ovtbls, &ovlock, &ovuser, &ovbuff,
                                 &not_found);
```

```
        } /* end while not_found and fgets != null */

        strcpy(scales[index].ds_name, "OVTBLS");
        dial_label = XmStringCreate(scales[index].ds_name,
                                    XmSTRING_DEFAULT_CHARSET);
        scales[index].ds_label = dial_label;
        scales[index].ds_style = STYLE_OVER;
        scales[index].ds_value = ovtbls;
        scales[index].ds_amount = ovtbls;
        scales[index].ds_redzone = 'n';
        if (scales[index].ds_amount >= ALARM_OVER)
           scales[index].ds_alarm = 'y';
        else
           scales[index].ds_alarm = 'n';

        index++;
        actual_max = index;

        strcpy(scales[index].ds_name, "OVLOCK");
        dial_label = XmStringCreate(scales[index].ds_name,
                                    XmSTRING_DEFAULT_CHARSET);
        scales[index].ds_label = dial_label;
        scales[index].ds_style = STYLE_OVER;
        scales[index].ds_value = ovlock;
        scales[index].ds_amount = ovlock;
        scales[index].ds_redzone = 'n';
        if (scales[index].ds_amount >= ALARM_OVER)
           scales[index].ds_alarm = 'y';
        else
           scales[index].ds_alarm = 'n';

        index++;
        actual_max = index;

        strcpy(scales[index].ds_name, "OVUSER");
        dial_label = XmStringCreate(scales[index].ds_name,
                                    XmSTRING_DEFAULT_CHARSET);
        scales[index].ds_label = dial_label;
        scales[index].ds_style = STYLE_OVER;
        scales[index].ds_value = ovuser;
        scales[index].ds_amount = ovuser;
        scales[index].ds_redzone = 'n';
        if (scales[index].ds_amount >= ALARM_OVER)
           scales[index].ds_alarm = 'y';
        else
           scales[index].ds_alarm = 'n';

        index++;
        actual_max = index;

        strcpy(scales[index].ds_name, "OVBUFF");
        dial_label = XmStringCreate(scales[index].ds_name,
                                    XmSTRING_DEFAULT_CHARSET);
        scales[index].ds_label = dial_label;
        scales[index].ds_style = STYLE_OVER;
```



```
        scales[index].ds_value = ovbuff;
        scales[index].ds_amount = ovbuff;
        scales[index].ds_redzone = 'n';
        if (scales[index].ds_amount >= ALARM_OVER)
           scales[index].ds_alarm = 'y';
        else
           scales[index].ds_alarm = 'n';

        index++;
        actual_max = index;

     } /* end else if for getting over values */

     /*********************************************************/
     /*** Find deadlk in profile.  Place into scales array. ***/
     /*********************************************************/
     else if ( strstr(config_str,"bufwaits") != NULL)
     {
          /* while loop locates deadlk value */
          not_found = 1;
          while (not_found && fgets(config_str, MAX, fp) != NULL)
          {
                  get_deadlk(config_str, &deadlk, &not_found);

          } /* end while not_found and fgets != null */

          strcpy(scales[index].ds_name, "DEADLK");
          dial_label = XmStringCreate(scales[index].ds_name,
                                      XmSTRING_DEFAULT_CHARSET);
          scales[index].ds_label = dial_label;
          scales[index].ds_style = STYLE_DEADLK;
          scales[index].ds_value = deadlk;
          scales[index].ds_amount = deadlk;
          scales[index].ds_redzone = 'n';
          if (scales[index].ds_amount >= ALARM_DEADLK)
          scales[index].ds_alarm = 'y';
          else
          scales[index].ds_alarm = 'n';

          index++;
          actual_max = index;


     } /* end else if for getting deadlk */

     else
          continue;

   } /* end while cont and fgets */

   *scale_index = actual_max;
   fclose(fp);
}
```

```
/*****************************************************************************/
/* mon_fcts.c -- Holds all function bodies used to gather info from tbstat-a */
/*               file.                                                       */
/*                                                                           */
/*      Eric B. Gronholz                                                     */
/*      Systems Migration, Inc.                                              */
/*      501 E. Hwy 13  Suite 112                                             */
/*      Burnsville, MN  55337                                                */
/*                                                                           */
/*      May, 1993                                                            */
/*****************************************************************************/
 #include "mon_defs.h"

/**********************/
/*** FIND CKPTINTVL ***/
/**********************/
double ckpt_intvl_min(fp)
        FILE *fp;
{
        char config_str[MAX];
        double min_intvl;       /* value holding CKPTINTVL */
        long count = 0;
        long not_found = 1;

        while (not_found && fgets(config_str, MAX, fp) != NULL)
        {
           if ( strstr(config_str, "CKPTINTVL") != NULL )
           {
                /* while loop locates position of min value for CKPTINTVL
                   in config_str by reading past any non-numbers in file */
                while (config_str[count] < '0' || config_str[count] > '9')
                {
                   count++;
                }
                min_intvl = atof(config_str+count);
                not_found = 0;
           }
        }

        return min_intvl;

}


/****************************/
/*** GET CHECKPOINT TIMES ***/
/****************************/
void get_ckpts(config_str, ckpt_array, ca_index)
        char config_str[];
        double ckpt_array[];
        long ca_index;
{
        double hours, minutes, seconds, total_time;
        long count = 0;         /* position in config_str[] */

        /* read past any preliminary blanks */
```



```
        while (config_str[count] == ' ')
           count++;

        /* find hours and convert into seconds */
        hours = atof(config_str+count);         /*change hours to double*/
        hours = hours * 3600;                   /* convert hours to seconds */

        /* read past hours and ready for minutes */
        while (config_str[count] != ':' )
           count++;
        while (config_str[count] == ':')
           count++;

        /* find minutes and convert into seconds */
        minutes = atof(config_str+count);       /*change minutes to double*/
        minutes = minutes * 60;                 /* convert minutes to seconds */

        /* read past minutes and ready for seconds */
        while (config_str[count] != ':' )
           count++;
        while (config_str[count] == ':')
           count++;

        /* find seconds */
        seconds = atof(config_str+count);       /*change seconds to double*/

        /* find total time in seconds and store in ckpt_array */
        total_time = (hours + minutes + seconds);
        ckpt_array[ca_index] = total_time;
}


/***********************************/
/*** COMPUTE CHECKPOINT INTERVALS ***/
/***********************************/
void compute_ckpt_intvl(ckpt_array, ca_index, min_ckpt_intvl,
                        avg_intvl, set_alarm)
        double  ckpt_array[];
        long    ca_index;
        double  min_ckpt_intvl;
        double  *avg_intvl;
        char    *set_alarm;

{
        static int below_ckpts = 0;
        double interval;
        double total_int;
        int i;

        /* Check to make sure there are at least 2 elements in ckpt_array. */
        /* If not, set avg_intvl to 0.  */
        if (ca_index >= 1)
        {
           for (i = 1; i < ca_index; i++)
           {
                interval = ckpt_array[i] - ckpt_array[i-1];
```

```
                total_int = total_int + interval;
            }

            *avg_intvl = ( total_int/ca_index);

            /*********************************************/
            /* Keep track of the number of times in a row */
            /* avg intvl is below half of CKPTINTVL.      */
            /*********************************************/
            if (*avg_intvl <= (min_ckpt_intvl/2))
            {
                below_ckpts++;
            }
            else
                below_ckpts = 0;
        }
        else
            *avg_intvl = 0.0;

        if (below_ckpts >= ALARM_CKPT)
            *set_alarm = 'y';

}  /* end compute_ckpt_intvl() */



/**********************************/
/*** GET ACTIVE AND TOTAL VALUES ***/
/**********************************/
void get_act_tot(config_str, active, total, not_found)
        char config_str[];
        double *active, *total;
        char *not_found;
{
        long count = 0;         /* position in config_str[] */

        if ( (strstr(config_str,"active") != NULL) ||
             (strstr(config_str,"modified") != NULL) )
        {
              *not_found = 0;

              /* Get active value */
              while (config_str[count] < '0'  || config_str[count] > '9') {
                 count++;
              };

              *active = atof(config_str+count); /*change active to double*/

              /* need to increment past active value */
              while (config_str[count] >= '0'  && config_str[count] <= '9') {
                 count++;
              };

              /* Get total value */
              while (config_str[count] < '0'  || config_str[count] > '9') {
                 count++;
```



```
              };
              *total = atof(config_str+count); /*change total value to double*/


        } /* end if active != null */

}  /* end get_act_tot function */



/*****************************/
/*** GET CHUNK INFORMATION ***/
/*****************************/
void get_chnk_info(config_str, chunk_percent, chunk_tot_percent)
        char config_str[];
        double *chunk_percent, *chunk_tot_percent;

{
        long count = 0;         /* position in config_str[] */
        long i;
        double chunk_size, pages_free;
        static double total_size, total_free;

        if (strstr(config_str,"active") == NULL)
        {
           /* Chunk size is the fifth set of info under chunk column headers.
              Must read past first four sets of info.  */

           for (i = 1; i <= CHUNK_SIZE_POS; i++)
           {
                /*  Read past characters and then read past blanks  */
                while (config_str[count] != ' ') {
                   count++;
                };
                while (config_str[count] == ' ') {
                   count++;
                };
           }

              /* Get size of chunk */
              chunk_size=atof(config_str+count); /*change chunk_size to double*/

              /* need to increment past active value */
              while (config_str[count] != ' ') {
                 count++;
              };

              /* Get number of pages free */
              while (config_str[count] == ' ') {
                 count++;
              };
              pages_free=atof(config_str+count); /*change pages_free to double*/

              /* Compute percent of chunk used */
              *chunk_percent = (chunk_size - pages_free)/(chunk_size);
```

```
            /* Compute total chunk size, total pages free, and total percent
               free */
            total_size = total_size + chunk_size;
            total_free = total_free + pages_free;
            *chunk_tot_percent = (total_size - total_free)/total_size;

        }; /* end if active == null */

}  /* end get_chnk_info function */


/****************************************/
/*** GET PHYSICAL AND LOGICAL LOGGINGS ***/
/****************************************/
void get_loggings(config_str, which_log, p_log_percent, l_log_percent,
                  not_found)
        char config_str[];
        int which_log;
        double *p_log_percent;
        double *l_log_percent;
        char *not_found;
{
        int i;
        int read_size, read_page;
        double buf_size, pages_io;
        long count = 0;

        *not_found = 0;

        if (which_log == 0)  /* physical log */
        {
           read_size = P_LOG_READ_SIZE_POS;
           read_page = P_LOG_READ_PAGE_POS;
        }
        else  /* logical log */
        {
           read_size = L_LOG_READ_SIZE_POS;
           read_page = L_LOG_READ_PAGE_POS;
        }

        /*** Get bufsize value ***/
        /* read past blanks, bufused, and bufsize -- unnecessary info */
        for (i = 1; i < read_size; i++)
        {
           while (config_str[count] == ' ')
                count++;
           while (config_str[count] != ' ')
                count++;
        }

        /* read past blanks before bufsize value and then get bufsize value */
        while (config_str[count] == ' ')
           count++;
        buf_size = atof(config_str+count); /* change pages_io to double */

        /* prepare for pages/io by reading past bufsize value */
```



```
        while (config_str[count] != ' ')
           count++;

        /*** Get pages/io ***/
        /* read past unnecessary info */
        for (i = read_size + 1; i < read_page; i++)
        {
           while (config_str[count] == ' ')
                count++;
           while (config_str[count] != ' ')
                count++;
        }

        /* read past blanks before pages/io value and then get bufsize value */
        while (config_str[count] == ' ')
           count++;

        pages_io = atof(config_str+count); /* change pages_io to double */

        if (which_log == 0)
           *p_log_percent = (pages_io/buf_size);
        else
           *l_log_percent = (pages_io/buf_size);


}  /* end loggings */


/*****************************************************************/
/*** GET BUFREADS % CACHED AND BUFWRITS % CACHED FROM PROFILE ***/
/*****************************************************************/
void get_percent_cached(config_str, rd_cache, wt_cache, not_found)
        char config_str[];
        double *rd_cache, *wt_cache;
        char *not_found;
{
        int i;
        long count = 0;

        *not_found = 0;

        /* The following for loop skips past values in the config_str until
           it reaches the first %cached.*/
        for (i = 1; i < READ_CACHE_POS; i++)
        {
           /* read past blanks */
           while (config_str[count] == ' ')
              count++;

           /* read past unnecessary values */
           while (config_str[count] != ' ')
              count++;

        } /* end for i loop */

        /* read past blanks before getting first %cache value*/
```

```
        while (config_str[count] == ' ')
           count++;

        *rd_cache = atof(config_str+count); /* change rd_cache to double */

        /* read past rd_cache value */
        while (config_str[count] != ' ')
           count++;

        /* The following for loop skips past values in the config_str until
           it reaches the second %cached.*/
        for (i = READ_CACHE_POS + 1; i < WRIT_CACHE_POS; i++)
        {
           /* read past blanks */
           while (config_str[count] == ' ')
              count++;

           /* read past unnecessary values */
           while (config_str[count] != ' ')
              count++;

        } /* end for i loop */

        /* read past blanks before getting second %cache value*/
        while (config_str[count] == ' ')
           count++;

        *wt_cache = atof(config_str+count); /* change wt_cache to double */

}  /* end get_percent_cached */


/***********************************/
/*** GET OVER VALUES FROM PROFILE ***/
/***********************************/
void get_over_vals(config_str, ovtbls, ovlock, ovuser, ovbuff, not_found)
        char config_str[];
        int *ovtbls, *ovlock, *ovuser, *ovbuff;
        char *not_found;
{
        int i;
        long count = 0;

        *not_found = 0;

        /* The following for loop reads the first four values in the config_str
           since ovbuff is the last value (at postion 4). */
        for (i = 1; i <= OVBUFF_POS; i++)
        {
           /* read past blanks before value and then get value as an integer. */
           while (config_str[count] == ' ')
              count++;

           if (i == 1)
           {
```



```
                *ovtbls = atoi(config_str+count); /* change ovtbls to integer */
           }
           else if (i == 2)
           {
                *ovlock = atoi(config_str+count); /* change ovtbls to integer */
           }
           else if (i == 3)
           {
                *ovuser = atoi(config_str+count); /* change ovtbls to integer */
           }
           else if (i == 4)
           {
                *ovbuff = atoi(config_str+count); /* change ovtbls to integer */
           };

           while (config_str[count] != ' ')
                count++;

        } /* end for i loop */

}  /* end get_over_vals */


/*****************************/
/*** GET DEADLK FROM PROFILE ***/
/*****************************/
void get_deadlk(config_str, deadlk, not_found)
        char config_str[];
        int *deadlk;
        char *not_found;
{
        int i;
        long count = 0;

        *not_found = 0;

        /* The following for loop skips past values in the config_str until
           it reaches the deadlk value. */
        for (i = 1; i <= DEADLK_POS; i++)
        {
           /* read past blanks */
           while (config_str[count] == ' ')
              count++;

           if (i == DEADLK_POS)
           {
                *deadlk = atoi(config_str+count); /* change deadlk to integer */
           }
           else
           {
                /* read past unnecessary values */
                while (config_str[count] != ' ')
                   count++;
           }
```

```
        } /* end for i loop */

} /* end get_deadlk */
```

```
/*****************************************************************/
/* "mon_interface.h" -- declarations of fcts and procedures      */
/*   used in get_display_info.c                                  */
/*                                                               */
/*      Eric B. Gronholz                                         */
/*      Systems Migration, Inc.                                  */
/*      501 E. Hwy 13  Suite 112                                 */
/*      Burnsville, MN  55337                                    */
/*                                                               */
/*      April/May 1993                                           */
/*****************************************************************/


#ifndef MONINTERFACE
#define MONINTERFACE

#include "mon_defs.h"

extern void get_display_info();
       /* get_display_info() is used to collect the information from the
          tbstat.info file and put it into the display_scales array of
          structs after it is converted into the correct format */

extern double ckpt_intvl_min();
       /* ckpt_intvl_min() is a function that returns the value of CKPTINVL in
          the tbstat-a file.  This is the minimum amount of time (in seconds)
          that the system should complete a checkpoint. */

extern void get_ckpts();
       /* reads times of checkpoints completed from tbstat-a file, converts the
          time into seconds, and stores those values in the ckpt_array.  This
          function returns the update ckpt_array and the current size of the
          array (ca_index).  */

extern void compute_ckpt_intvl();
       /* determines sizes of intervals between checkpoint times and then
          calculates the avg interval for all times in the ckpt_array.  It
          returns the avg total interval as well as the number of times that
          interval was below the CKPTINTVL value in the tbstat-a file. */

extern void get_act_tot();
       /* get_act_tot() is a function that retrieves the active and total
          values for a given item in the tbstat-a file.  Once a value is
          retrieved, it must be converted to a double floating point value
          because it is retrieved as a character value.  The function receives
          the string containing the active and total values; it returns those
          values as doubles and also returns not_found = 0 (to stop the while
          loop).  */

extern void get_chnk_info();
       /* get_chnk_info() is a function that retrieves the total number of
          pages in a given chunk (under the heading 'size' in the Chunks
          section of tbstat-a) and also the number of pages available in that
          chunk (free).  Once the values are retrieved, they must be converted
          to double floating point values because they are retrieved as a
          character values.  After the values are converted to doubles,
```



```
          get_chnk_info converts the number of free pages into the number of
          used pages (size - free) and then divides that number by the size of
          that particular chunk.  The function returns the percent of pages
          used in the chunk.  */

extern void get_loggings();
       /* gets physical and logical loggin info.  Returns (pages/io)/bufsize
          for both types of loggings.  */

extern void get_percent_cached();
       /* gets bufreads' %cached and bufwrits' %cached from Profile.  Returns
          both percentages. */

extern void get_over_vals();
       /* gets values for ovtbls, ovlock, ovuser, ovbuff in the Profile sectior
          of tbstat-a file */

extern void get_deadlk();
       /* gets and returns deaklk value from Profile. */

#endif
```

```
/*************************************************************************/
/* mon_defs.h  (Monitor definitions)                                     */
/*                                                                       */
/*  This file defines all of the global variables for dial.c,            */
/*  get_display_info.c, mon_fcts.c, and tbstat.c                         */
/*                                                                       */
/*      Eric B. Gronholz                                                 */
/*      Systems Migration, Inc.                                          */
/*      501 E. Hwy 13  Suite 112                                         */
/*      Burnsville, MN  55337                                            */
/*                                                                       */
/*      April/May 1993                                                   */
/*************************************************************************/


#ifndef MONDEFS
#define MONDEFS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>

#define MAX             133     /* max length of a line */
#define MAX_SCALES       25     /* max length of scales array */
#define MAX_CKPTS        20     /* max length of ckpt_array */
#define MAX_CHUNKS       10     /* max length of chunks array */
#define MAX_FILES        20     /* number of tbstat files used for demo */
#define MAX_LINES        30
#define MAX_ARGS         10

/** the following geometry constants should be changed
    if resources are changed */
#define COLUMNS    5            /* # of columns displayed */
#define ROWS       4            /* # of rows displayed */
#define T_WIDTH  850            /* width of row */
#define T_HEIGHT 560            /* height of column */
#define WIDTH    200            /* width of alarm window */
#define HEIGHT   150            /* height of alarm window */

/* constants for items to be displayed */

#define STYLE_CKPT      "dial"
#define RED_CKPT         50
#define ALARM_CKPT       5

#define STYLE_USER      "dial"
#define RED_USER         0
#define USERS_REMAIN     5

#define STYLE_TRANS     "dial"
#define RED_TRANS        85
#define ALARM_TRANS      95
```



```
#define STYLE_LOCKS     "dial"
#define RED_LOCKS        85
#define ALARM_LOCKS      95

#define STYLE_BUFFS     "dial"
#define RED_BUFFS        85
#define ALARM_BUFFS      95

#define STYLE_TBLS      "dial"
#define RED_TBLS         85
#define ALARM_TBLS       95

#define STYLE_CHUNKS    "dial"
#define CHUNK_SIZE_POS   4
#define RED_CHUNKS       90
#define ALARM_CHUNKS     90

#define P_LOG_READ_SIZE_POS 3
#define P_LOG_READ_PAGE_POS 6
#define L_LOG_READ_SIZE_POS 3
#define L_LOG_READ_PAGE_POS 8
#define STYLE_LOG       "dial"
#define RED_LOG          75
#define ALARM_LOG        75

#define READ_CACHE_POS   4
#define WRIT_CACHE_POS   8
#define STYLE_CACHE     "dial"
#define RED_RD_CACHE     95
#define RED_WT_CACHE     85
#define ALARM_RD_CACHE   95
#define ALARM_WT_CACHE   85

#define OVBUFF_POS       4
#define STYLE_OVER      "alarmwin"
#define RED_OVER         0
#define ALARM_OVER       1

#define DEADLK_POS       4
#define STYLE_DEADLK    "alarmwin"
#define RED_DEADLK       0
#define ALARM_DEADLK     1


struct display_scale {
   char ds_name[MAX];   /* item in tbstat, i.e. "Users" */
   XmString ds_label;   /* same as ds_name.  Used to label dial */
   char *ds_style;      /* "dial" or "alarmwin" */
   double ds_value;     /* actual value of item to be displayed */
   int ds_amount;       /* height or distance displayed on meter or dial */
   char ds_redzone;     /* has dial hit its redzone?  'y' or 'n' */
   char ds_alarm;       /* sound alarm?  'y' or 'n' */
 };


struct update_info {
```

```
    Widget dial_widg[MAX_SCALES];
    Widget label_widg[MAX_SCALES];
    struct display_scale dials[MAX_SCALES];
    long d_index;
};
struct update_info upd_info;


#endif
```

```
/*************************************************************************/
/*  Makefile -- used to compile dial.c, get_display_info.c, and mon_fcts.c */
/*                                                                       */
/*      Eric B. Gronholz                                                 */
/*      Systems Migration, Inc.                                          */
/*      501 E. Hwy 13  Suite 112                                         */
/*      Burnsville, MN  55337                                            */
/*                                                                       */
/*      April/May 1993                                                   */
/*************************************************************************/

CC = cc
XSLIBDIR = ~/Motif/lib
DEBUG =
# HP Machines
CFLAGS =  $(DEBUG) -I$(XSLIBDIR)
# DEC and Sun
#CFLAGS = $(DEBUG) -I$(XSLIBDIR)

LIBS = ~/Motif/lib/libXs.a -lXm -lXt -lX11 -lm

.o:.c

all: lib dial

lib : Dial.o
        ar ruv ~/Motif/lib/libXs.a Dial.o

dial: dial.o get_display_info.o mon_fcts.o
        $(CC) $(CFLAGS) -o dial dial.o get_display_info.o mon_fcts.o $(LIBS)

dial.o: dial.c mon_defs.h mon_interface.h
        cc -c dial.c $(LIBS)

get_display_info.o      :  get_display_info.c mon_defs.h mon_interface.h
        cc -c get_display_info.c

mon_fcts.o      :  mon_fcts.c mon_defs.h
        cc -c mon_fcts.c

clean:
        rm *.o
```

```
/*****************************************************************************
 *  FILENAME:  tbstat.c
 *  AUTHOR  :  Eric B. Gronholz
 *             Systems Migration, Inc.
 *             501 E. Hwy 13 Suite 112
 *             Burnsville, MN  55337
 *
 *             April/May 1993
 *
 * PROGRAM DESCRIPTION: tbstat is a program simulating the tbstat command
 *                      for the INFORMIX OnLine Database Server.  The program's
 *                      primary purpose is to retrieve the correct tbstat data
 *                      file created for demostrating the X Window Application
 *                      "Monitor" on systems which do not have INFORMIX
 *                      installed on them.  Since systems without INFORMIX
 *                      cannot run the actual tbstat command, this program
 *                      retrieves one of the files containing tbstat
 *                      information and displays it to the screen.  Files
 *                      containing this information are named "tbstat1",
 *                      "tbstat2", "tbstat3", etc.  Each of these files
 *                      contains slightly different data in order to fully
 *                      utilize all of the tools "Monitor" has to offer.
 *                      The information is collected by the Monitor program
 *                      and displayed graphically.
 *****************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mon_defs.h"

#define STRING_SIZE 100

int main() {
        int count;
        char count_string[STRING_SIZE];
        char file_string[STRING_SIZE] =
                "/usr/people/students/cs/ebgronho/thesis/tbstat_files/tbstat";

        FILE *fp;       /* file pointer for "file_count" file.  Use "file_count"
                           to get number of next tbstat file.  File initially
                           contains the number 0, which is retrieved into this
                           program, incremented, then concatenated with
                           file_string "tbstat" to create a new file_string
                           "tbstat1."  The incremented value is written back
                           into the "file_count" file to allow for the formation
                           of the next filename, i.e. "tbstat2", and so on.
                        */

        char *comm = "cat";             /* command to "cat" a tbstat# file */
        char command[STRING_SIZE];      /* holds command to be executed by the
                                           system function. */

        if ((fp = fopen("count_file", "r+")) == NULL) {
           printf ("Can't find file called \"count_file\"\n");
           exit(1);
```



```
        };

        /*
           get number from count_file and increment.  Write new count to count_
           file for next run of tbstat.  Incremented number is the number of
           current tbstat file to use.
        */
        fscanf(fp, "%d", &count);
        /*
          if count = MAX_FILES, reset count_file to 0 and start over
          if count < MAX_FILES, find another tbstat file.
        */


        if (count == MAX_FILES)
           count = 0;
        if (count < MAX_FILES) {
           count++;
           fseek(fp, 0L, SEEK_SET);
           fprintf(fp, "%d", count);
           sprintf(count_string, "%d", count);
           strcat(file_string, count_string);
           sprintf(command, "%s %s", comm, file_string);
           system(command);
        }
        fclose(fp);
        return 0;
}
```

```
/*************************************************************************/
/*  alarm.c : create alarms for selected tbstat parameters               */
/*      By:                                                              */
/*              Eric B. Gronholz                                         */
/*              Systems Migration, Inc.                                  */
/*              501 E. Hwy 13  Suite 112                                 */
/*              Burnsville, MN  55337                                    */
/*                                                                       */
/*              April/May, 1993                                          */
/*                                                                       */
/*  This program receives a command line message argument which it prints */
/*  in a memo widget it creates.  The window includes a close button that */
/*  the user may click on to destroy the memo.                           */
/*************************************************************************/

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/MessageB.h>
#include <X11/Shell.h>
#include "/usr/people/students/cs/ebgronho/Motif/lib/libXs.h"


#define MAX_ARGS  10
#define MAX       133

void ok_callback();
void help_callback();
XmString xs_str_array_to_xmstr(char **,int);

Widget  alarm, toplevel, dialog;
XmString        alarm_msg;
Arg             alarm_args[20];
int             n, i, p;
char            need_help;

int main (int argc, char ** argv)

{
   char  **mess;

   char *chkpts[] = {
    "Average interval between checkpoints",
    "is under 50% of specified CKPTINTVL",
    "in tbstat configuration file",
    "",
    "The system is been backing itself",
    "up too frequently.  This is causing",
    "excess use of disk and suspension of",
    "current processes whichs slows",
    "down the system",
    "", ""};
   char *users[] = {
    "Number of users within 5 of",
    "specified USERS allowed in",
    "tbstat configuration file",
```



```
    "",
    "May need to kill certain processes",
    "or limit the number of users",
    "", ""};
   char *trans[] = {
    "Number of transactions within 95%",
    "of specified TRANSACTIONS total in ",
    "tbstat configuration file",
    "",
    "User activity is high.  May need to",
    "kill some processes",
    "","",};
   char *locks[] = {
    "Number of locks within 95% of",
    "specified LOCKS total in",
    "tbstat configuration file",
    "",
    "Check to see how necessary these",
    "locks are.  Eliminate those that",
    "are not essential",
    "", ""};
  char *buffs[] = {
    "Number of modified buffers within",
    "95% of tbstat specified BUFFERS",
    "total in configuration file",
    "",
    "May need to increase number of",
    "buffers allowed by the system",
    "", ""};
   char *tbls[] = {
    "Number of tblspaces within 95%",
    "of specified TBLSPACES total",
    "in tbstat configuration file",
    "",
    "May need to allocate more space",
    "on the system for the database",
    "", ""};
   char *chunk[] = {
    "Total chunk memory",
    "is over 90% full",
    "",
    "Memory is nearing capacity.",
    "Need to allocate more memory",
    "on system to database",
    "", ""};
   char *p_log[] = {
    "Physical logging pages_io/bufsize",
    "is under 75%",
    "",
    "Available buffer space is not being",
    "used fully.  Decrease the buffer",
    "size to increase efficiency",
    "", ""};
   char *l_log[] = {
    "Logical logging pages_io/bufsize",
    "is under 75%",
```

```
    "",
    "Available buffer space is not being",
    "used fully.  Decrease the buffer",
    "size to increase efficiency",
    "", ""};
   char *r_cache[] = {
    "Percent cached for dskreads,",
    "pagreads, and bufreads",
    "is under 95%",
    "",
    "Try increasing the number of",
    "of buffers to increase performance",
    "",""};
   char *w_cache[] = {
    "Percent cached for dskwrits,",
    "pagwrits, and bufwrits",
    "is under 85%",
    "",
    "Try increasing the number of",
    "of buffers to increase performance",
    "",""};
   char *ovtbls[] = {
    "OVTBLS value has been set",
    "",
    "Database attempted to exceed total",
    "number of available tblspaces.",
    "May need to allocate more space",
    "on the system for the database",
    "", ""};
   char *ovlock[] = {
    "OVLOCK value has been set",
    "",
    "Database attempted to exceed total",
    "number of available locks.",
    "Check to see how necessary these",
    "locks are.  Eliminate those that",
    "are not essential",
    "", ""};
   char *ovuser[] = {
    "OVUSER value has been set",
    "",
    "Database attempted to exceed total",
    "number of available users.",
    "May need to kill certain processes",
    "or limit the number of users",
    "", ""};
   char *ovbuff[] = {
    "OVBUFF value has been set",
    "",
    "Database attempted to exceed total",
    "number of available buffers.",
    "May need to increase number of",
    "buffers allowed by the system",
    "", ""};
   char *deadlk[] = {
    "DEADLK value has been set",
```



```
    "",
    "A potential deaklock was dedected",
    "and prevented.",
    "", ""};

        char *msg_help_ckpts[] = {"CHECKPOINTS shows the times when",
                                  "the system was last flushed to disk.",
                                  "Dial shows average of these times.",
                                  "", ""};
        char *msg_help_users[] = {"USERS shows all users currently",
                                  "accessing the OnLine system.  Dial",
                                  "shows number of current user processes",
                                  "as a percentage of total user processes",
                                  "allowed.",
                                  "", ""};
        char *msg_help_trans[] = {"TRANSACTS shows all the transaction",
                                 "activity associated with current users.",
                                 "Dial shows active transactions as a",
                                 "percentage of total transactions allowed.",
                                 "", ""};
        char *msg_help_locks[] = {"LOCKS shows all active locks in the",
                                 "OnLine system.  Dial shows active locks",
                                 "as a percentage of total locks allowed.",
                                 "", ""};
        char *msg_help_buffs[] = {"BUFFERS shows the state of all buffers",
                                 "currently being modified or latched by",
                                 "database server processes.  Dial shows",
                                 "buffers modified as a percentage of the",
                                 "total buffers allowed.",
                                 "",""};
        char *msg_help_tbls[]  = {"TBLSPACES show all active tblspaces in",
                                 "the OnLine system.  Dial shows active",
                                 "tblspaces as a percentage of total",
                                 "tblspaces allowed.",
                                 "",""};
        char *msg_help_chnks[] = {"CHUNK shows the component chunks that",
                                 "make up the dbspaces and blobspaces.",
                                 "Dial shows number of pages used as a",
                                 "percentage of total pages available for",
                                 "a given chunk.",
                                 "",""};
        char *msg_help_inact[] ={"INACTIVE indicates that there are more",
                                 "chunks available than are currently being",
                                 "used.  Dial is dormat while names INACTIVE,",
                                 "but displays chunk information if it is has",
                                 "the name of a chunk.",
                                 "",""};
        char *msg_help_plog[]  = {"PHYS_LOG shows information about how",
                                 "much the physical log buffer is being",
                                 "used.  Dial shows pages/io--calculated as",
                                 "(numpages/numwrites)--as a percentage of",
                                 "the size of the buffer.",
                                 "",""};
        char *msg_help_llog[]  = {"LOGIC_LOG shows information the logical",
                                 "logs and the logical log buffer.  Dial",
                                 "shows pages/io--calculated as",
```

```
                                "(numpages/numwrites)--as a percentage of",
                                "the size of the buffer.",
                                "","};
        char *msg_help_rcache[]={"R_CACHED is a measure of the number of",
                                "reads from buffers compared to reads from",
                                "disk.  Dial shows the percentage of reads",
                                "cached--calculated as (bufreads-dskreads)/",
                                "bufreads.",
                                "","};
        char *msg_help_wcache[]={"W_CACHED is a measure of the number of",
                                "writes to buffers compared to writes to",
                                "disk.  Dial shows the percentage of writes",
                                "cached--calculated as (bufwrits-dskwrits)/",
                                "bufwrits.",
                                "","};


     /* check for the argument.  If help does not preceed the name of
        the dial in the argument list, find a name's message; otherwise,
        find name's help message. */
   if (strcmp(argv[argc-2], "help") != 0)
   {
      need_help = 'n';
      if (strcmp(argv[argc-1], "CHECKPOINTS") == 0) {
         mess = chkpts;
      }
      else if (strcmp(argv[argc-1], "USERS") == 0) {
         mess = users;
      }
      else if (strcmp(argv[argc-1], "TRANSACTS") == 0) {
         mess = trans;
      }
      else if (strcmp(argv[argc-1], "LOCKS") == 0) {
         mess = locks;
      }
      else if (strcmp(argv[argc-1], "BUFFERS") == 0) {
         mess = buffs;
      }
      else if (strcmp(argv[argc-1], "TBLSPACES") == 0) {
         mess = tbls;
      }
      else if ((strstr(argv[argc-1], "CHUNK") != 0) ||
               (strcmp(argv[argc-1], "INACTIVE") == 0))
      {
         mess = chunk;
      }
      else if (strcmp(argv[argc-1], "PHYS_LOG") == 0) {
         mess = p_log;
      }
      else if (strcmp(argv[argc-1], "LOGIC_LOG") == 0) {
         mess = l_log;
      }
      else if (strcmp(argv[argc-1], "READS_CACHED") == 0) {
         mess = r_cache;
      }
      else if (strcmp(argv[argc-1], "WRITES_CACHED") == 0) {
```



```
         mess = w_cache;
      }
      else if (strcmp(argv[argc-1], "OVTBLS") == 0) {
         mess = ovtbls;
      }
      else if (strcmp(argv[argc-1], "OVLOCK") == 0) {
         mess = ovlock;
      }
      else if (strcmp(argv[argc-1], "OVUSER") == 0)
      {
         mess = ovuser;
      }
      else if (strcmp(argv[argc-1], "OVBUFF") == 0) {
         mess = ovbuff;
      }
      else if (strcmp(argv[argc-1], "DEADLK") == 0) {
         mess = deadlk;
      }
   }
   else
   {
      need_help = 'y';
      if (strcmp(argv[argc-1], "CHECKPOINTS") == 0) {
         mess = msg_help_ckpts;
      }
      else if (strcmp(argv[argc-1], "USERS") == 0) {
         mess = msg_help_users;
      }
      else if (strcmp(argv[argc-1], "TRANSACTS") == 0) {
         mess = msg_help_trans;
      }
      else if (strcmp(argv[argc-1], "LOCKS") == 0) {
         mess = msg_help_locks;
      }
      else if (strcmp(argv[argc-1], "BUFFERS") == 0) {
         mess = msg_help_buffs;
      }
      else if (strcmp(argv[argc-1], "TBLSPACES") == 0) {
         mess = msg_help_tbls;
      }
      else if (strstr(argv[argc-1], "CHUNK") != 0) {
         mess = msg_help_chnks;
      }
      else if (strcmp(argv[argc-1], "INACTIVE") == 0) {
         mess = msg_help_inact;
      }
      else if (strcmp(argv[argc-1], "PHYS_LOG") == 0) {
         mess = msg_help_plog;
      }
      else if (strcmp(argv[argc-1], "LOGIC_LOG") == 0) {
         mess = msg_help_llog;
      }
      else if (strcmp(argv[argc-1], "READS_CACHED") == 0) {
         mess = msg_help_rcache;
      }
      else if (strcmp(argv[argc-1], "WRITES_CACHED") == 0) {
```

```
         mess = msg_help_wcache;
      }

   } /* end if argv == "help" */

      toplevel = XtInitialize(argv[0], "Alarm", NULL, 0,
                              &argc, argv);

      help_callback(NULL, mess, NULL);

      XtRealizeWidget(dialog);
      XtMainLoop();


}

/*****************************************************************************/
/*****************************************************************************/
void ok_callback( w,  client_data,  call_data)
   Widget               w;
   caddr_t              client_data;
   XmAnyCallbackStruct  *call_data;
{
   exit(1);
}
/*****************************************************************************/
void help_callback( w,  mess,  call_data)
   Widget               w;
   char                 *mess[];
   XmAnyCallbackStruct  *call_data;
{
   Widget label;

      /* Create warning dialog to display tbstat warnings */
      n = 0;
      XtSetArg(alarm_args[n], XmNautoUnmanage, FALSE); n++;
      if (need_help == 'n')
         dialog = XmCreateMessageDialog(toplevel, "WARNING", alarm_args, n);
      else
         dialog = XmCreateMessageDialog(toplevel, "HELP", alarm_args, n);

      XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_CANCEL_BUTTON));

      /* ok_callback closes the warning window */
      XtAddCallback(dialog, XmNokCallback, ok_callback, NULL);

      /* Get next message from mess array--read until a null is read */
      for (i=0; mess[i][0] != '\0'; i++);

      /* Make text left justified */
      label = XmMessageBoxGetChild(dialog, XmDIALOG_MESSAGE_LABEL);
      n = 0;
      XtSetArg(alarm_args[n], XmNalignment, XmALIGNMENT_BEGINNING); n++;
      XtSetValues(label, alarm_args, n);

      /* change array of character strings into an XmString */
```



```
      alarm_msg = xs_str_array_to_xmstr(mess, i);

      /* display warning message to warning window */
      n = 0;
      XtSetArg(alarm_args[n], XmNmessageString, alarm_msg); n++;
      XtSetValues(dialog, alarm_args, n);

      /* Read for next message.  If message is empty, messages complete */
      if (mess[++i][0] == '\0')
        XtUnmanageChild(XmMessageBoxGetChild(dialog, XmDIALOG_HELP_BUTTON));
      else
        XtAddCallback(dialog, XmNhelpCallback, help_callback, &mess[i]);

      XtManageChild(dialog);

}
```

```
/*************************************************************************/
/*  Make_alarm -- used to compile alarm.c                                */
/*                                                                       */
/*      Eric B. Gronholz                                                 */
/*      Systems Migration, Inc.                                          */
/*      501 E. Hwy 13  Suite 112                                         */
/*      Burnsville, MN  55337                                            */
/*                                                                       */
/*      April/May 1993                                                   */
/*************************************************************************/

CC = cc
XSLIBDIR = ~/Motif/lib
DEBUG =
# HP Machines
CFLAGS = $(DEBUG) -I$(XSLIBDIR)
# DEC and Sun
#CFLAGS = $(DEBUG) -I$(XSLIBDIR)

LIBS = ~/Motif/lib/libXs.a -lXm -lXt -lX11 -lm

.o:.c


alarm: alarm.o
        $(CC) $(CFLAGS) -o alarm alarm.o $(LIBS)

alarm.o:        alarm.c
        cc -c alarm.c $(LIBS)

clean:
        rm *.o
```

```
/*****************************************************************************
 * Dial.c: The Dial Widget
 *         From:
 *                  The X Window System,
 *            Programming and Applications with Xt
 *                  OSF/Motif Edition
 *         by
 *               Douglas Young
 *             Prentice Hall, 1990
 *
 *                  Example described on pages:  344-362
 *
 *
 *  Copyright 1989 by Prentice Hall
 *  All Rights Reserved
 *
 * This code is based on the OSF/Motif widget set and the X Window System
 *
 * Permission to use, copy, modify, and distribute this software for
 * any purpose and without fee is hereby granted, provided that the above
 * copyright notice appear in all copies and that both the copyright notice
 * and this permission notice appear in supporting documentation.
 *
 * Prentice Hall and the author disclaim all warranties with regard to
 * this software, including all implied warranties of merchantability and fitne
 * In no event shall Prentice Hall or the author be liable for any special,
 * indirect or cosequential damages or any damages whatsoever resulting from
 * loss of use, data or profits, whether in an action of contract, negligence
 * or other tortious action, arising out of or in connection with the use
 * or performance of this software.
 *
 * Open Software Foundation is a trademark of The Open Software Foundation, Inc
 * OSF is a trademark of Open Software Foundation, Inc.
 * OSF/Motif is a trademark of Open Software Foundation, Inc.
 * Motif is a trademark of Open Software Foundation, Inc.
 * DEC is a registered trademark of Digital Equipment Corporation
 * HP is a registered trademark of the Hewlett Packard Company
 * DIGITAL is a registered trademark of Digital Equipment Corporation
 * X Window System is a trademark of the Massachusetts Institute of Technology
 *****************************************************************************/


#include <stdio.h>
#include <math.h>
#include <X11/IntrinsicP.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/CoreP.h>
#include "DialP.h"
#include "Dial.h"

#define  RADIANS(x)  (M_PI * 2.0 * (x) / 360.0)
#define  DEGREES(x)  ((x) / (M_PI * 2.0) * 360.0)
#define  MIN_ANGLE   225.0
#define  MAX_ANGLE   270.0
#define  MIN(a,b)    (((a) < (b)) ? (a) :  (b))
```



```
static void    select_dial ();
static void    Initialize();
static void    Redisplay();
static void    Resize();
static void    Destroy();
static Boolean SetValues();

static char defaultTranslations[] = "<Btn1Down>: select()";

static XtActionsRec actionsList[] = {
  { "select",   (XtActionProc) select_dial},
};

static XtResource resources[] = {
  {XtNmarkers, XtCMarkers, XtRInt, sizeof (int),
    XtOffset(XsDialWidget, dial.markers), XtRString, "10"  },
  {XtNminimum, XtCMin, XtRInt, sizeof (int),
    XtOffset(XsDialWidget, dial.minimum), XtRString, "0"   },
  {XtNmaximum, XtCMax, XtRInt, sizeof (int),
    XtOffset(XsDialWidget, dial.maximum), XtRString, "100" },
  {XtNindicatorColor, XtCColor, XtRPixel, sizeof (Pixel),
    XtOffset(XsDialWidget, dial.indicator_color),
    XtRString, "Black"                                     },
  {XtNposition, XtCPosition, XtRPosition, sizeof (Position),
    XtOffset(XsDialWidget, dial.position), XtRString, "0"  },
  {XtNmarkerLength,XtCLength,XtRDimension,sizeof (Dimension),
    XtOffset(XsDialWidget, dial.marker_length),
    XtRString, "5"                                         },
  {XtNforeground, XtCForeground, XtRPixel, sizeof (Pixel),
    XtOffset(XsDialWidget, dial.foreground),
    XtRString, "Black"                                     },
  {XtNselectCallback,XtCCallback,XtRCallback,sizeof(caddr_t),
    XtOffset (XsDialWidget, dial.select), XtRCallback, NULL },
 };

XsDialClassRec  XsdialClassRec = {
     /* CoreClassPart */
  {
   (WidgetClass) &widgetClassRec,  /* superclass            */
   "Dial",                         /* class_name            */
   sizeof(XsDialRec),              /* widget_size           */
   NULL,                           /* class_initialize      */
   NULL,                           /* class_part_initialize */
   FALSE,                          /* class_inited          */
   Initialize,                     /* initialize            */
   NULL,                           /* initialize_hook       */
   XtInheritRealize,               /* realize               */
   actionsList,                    /* actions               */
   XtNumber(actionsList),          /* num_actions           */
   resources,                      /* resources             */
   XtNumber(resources),            /* num_resources         */
   NULLQUARK,                      /* xrm_class             */
   TRUE,                           /* compress_motion       */
   TRUE,                           /* compress_exposure     */
   TRUE,                           /* compress_enterleave   */
```

```
    TRUE,                           /* visible_interest      */
    Destroy,                        /* destroy               */
    Resize,                         /* resize                */
    Redisplay,                      /* expose                */
    SetValues,                      /* set_values            */
    NULL,                           /* set_values_hook       */
    XtInheritSetValuesAlmost,       /* set_values_almost     */
    NULL,                           /* get_values_hook       */
    NULL,                           /* accept_focus          */
    XtVersion,                      /* version               */
    NULL,                           /* callback private      */
    defaultTranslations,            /* tm_table              */
    NULL,                           /* query_geometry        */
    NULL,                           /* display_accelerator   */
    NULL,                           /* extension             */
    },
       /* Dial class fields */
  {
    0,                              /* ignore                */
    }
};

WidgetClass XsdialWidgetClass = (WidgetClass) &XsdialClassRec;

static void Initialize (request, new)
    XsDialWidget request, new;
{
  XGCValues values;
  XtGCMask  valueMask;
  /*
   * Make sure the window size is not zero. The Core
   * Initialize() method doesn't do this.
   */
  if (request->core.width == 0)
    new->core.width = 100;
  if (request->core.height == 0)
    new->core.height = 100;
  /*
   * Make sure the min and max dial settings are valid.
   */
  if (new->dial.minimum >= new->dial.maximum) {
    XtWarning ("Maximum must be greater than the Minimum");
    new->dial.minimum = new->dial.maximum - 1;
  }
  if (new->dial.position > new->dial.maximum) {
    XtWarning ("Position exceeds the Dial Maximum");
    new->dial.position =  new->dial.maximum;
  }
  if (new->dial.position < new->dial.minimum) {
    XtWarning ("Position is less than the Minimum");
    new->dial.position =  new->dial.minimum;
  }
  /*
   * Allow only MAXSEGMENTS / 2 markers
   */
  if(new->dial.markers > MAXSEGMENTS / 2){
```



```
    XtWarning ("Too many markers");
    new->dial.markers = MAXSEGMENTS / 2;
  }
  /*
   * Create the graphics contexts used for the dial face
   * and the indicator.
   */
  valueMask = GCForeground | GCBackground;
  values.foreground = new->dial.foreground;
  values.background = new->core.background_pixel;
  new->dial.dial_GC = XtGetGC (new, valueMask, &values);

  values.foreground = new->dial.indicator_color;
  new->dial.indicator_GC = XtGetGC (new, valueMask,&values);

  valueMask = GCForeground | GCBackground;
  values.foreground = new->core.background_pixel;
  values.background = new->dial.indicator_color;
  new->dial.inverse_GC = XtGetGC (new, valueMask, &values);

  Resize (new);
}

static void Destroy (w)
     XsDialWidget w;
{
  XtReleaseGC (w, w->dial.indicator_GC);
  XtReleaseGC (w, w->dial.inverse_GC);
  XtReleaseGC (w, w->dial.dial_GC);
  XtRemoveAllCallbacks (w, XtNselectCallback, w -> dial.select);
}


static calculate_indicator_pos(w)
     XsDialWidget w;
{
  double   normalized_pos, angle;
  Position indicator_length;
  /*
   * Make the indicator two pixels shorter than the
   * inner edge of the markers.
   */
  indicator_length=w->dial.outer_diam-w->dial.marker_length-2;
  /*
   * Normalize the indicator position to lie between zero
   * and 1, and then convert it to an angle.
   */
  normalized_pos = (w->dial.position - w->dial.minimum)/
                (float)(w->dial.maximum - w->dial.minimum);
  angle = RADIANS(MIN_ANGLE + MAX_ANGLE  * normalized_pos);
   /*
    * Find the x,y coordinates of the tip of the indicator.
    */
   w->dial.indicator_x = w->dial.center_x +
                                 indicator_length * sin(angle);
   w->dial.indicator_y = w->dial.center_y -
```

```
                                indicator_length  * cos(angle);
}

static void Resize (w)
    XsDialWidget w;
{
  double    angle, cosine, sine, increment;
  int       i;
  XPoint   *ptr;
  /*
   * Get the address of the first line segment.
   */
  ptr = w->dial.segments;
  /*
   * calculate the center of the widget
   */
  w->dial.center_x = w->core.width/2;
  w->dial.center_y = w->core.height/2;
  /*
   *  Generate the segment array containing the
   *  face of the dial.
   */
  increment = RADIANS(MAX_ANGLE) /(float)(w->dial.markers -1);
  w->dial.outer_diam = MIN(w->core.width, w->core.height) / 2;
  w->dial.inner_diam=w->dial.outer_diam-w->dial.marker_length;
  angle = RADIANS(MIN_ANGLE);

  for (i = 0; i < w->dial.markers;i++){
    cosine = cos(angle);
    sine   = sin(angle);
    ptr->x   = w->dial.center_x + w->dial.outer_diam * sine;
    ptr++->y = w->dial.center_y - w->dial.outer_diam * cosine;
    ptr->x   = w->dial.center_x + w->dial.inner_diam * sine;
    ptr++->y = w->dial.center_y - w->dial.inner_diam * cosine;
    angle += increment;
  }
 calculate_indicator_pos(w);
}

static void Redisplay (w, event, region)
     XsDialWidget  w;
     XEvent       *event;
     Region        region;
{
  if(w->core.visible){
    /*
     * Draw the markers used for the dial face.
     */
    XDrawSegments(XtDisplay(w), XtWindow(w),
                  w->dial.dial_GC,
                  w->dial.segments,
                  w->dial.markers);
    /*
     * Draw the indicator at its current position.
     */
    XDrawLine(XtDisplay(w), XtWindow(w),
```



```
              w->dial.indicator_GC,
              w->dial.center_x,
              w->dial.center_y,
              w->dial.indicator_x,
              w->dial.indicator_y);
    }
 }

static Boolean SetValues (current, request, new)
     XsDialWidget current, request, new;
{
  XGCValues  values;
  XtGCMask   valueMask;
  Boolean    redraw = FALSE;
  Boolean    redraw_indicator = FALSE;
  /*
   * Make sure the new dial values are reasonable.
   */
  if (new->dial.minimum >= new->dial.maximum) {
    XtWarning ("Minimum must be less than Maximum");
    new->dial.minimum = 0;
    new->dial.maximum = 100;
  }
  if (new->dial.position > new->dial.maximum) {
    XtWarning("Dial position is greater than the Maximum");
    new->dial.position = new->dial.maximum;
  }
  if (new->dial.position < new->dial.minimum) {
    XtWarning("Dial position is less than the Minimum");
    new->dial.position = new->dial.minimum;
  }
  /*
   * If the indicator color or background color
   * has changed, generate the GC's.
   */
 if(new->dial.indicator_color!=current->dial.indicator_color||
  new->core.background_pixel !=current->core.background_pixel){
    valueMask = GCForeground | GCBackground;
    values.foreground = new->dial.indicator_color;
    values.background = new->core.background_pixel;
    XtReleaseGC(new, new->dial.indicator_GC);
    new->dial.indicator_GC = XtGetGC(new, valueMask,&values);
    values.foreground = new->core.background_pixel;
    values.background = new->dial.indicator_color;
    XtReleaseGC(new, new->dial.inverse_GC);
    new->dial.inverse_GC = XtGetGC(new, valueMask, &values);
    redraw_indicator = TRUE;
  }
  /*
   * If the marker color has changed, generate the GC.
   */
  if (new->dial.foreground != current->dial.foreground){
    valueMask = GCForeground | GCBackground;
    values.foreground = new->dial.foreground;
    values.background = new->core.background_pixel;
    XtReleaseGC(new, new->dial.dial_GC);
```

```
    new->dial.dial_GC = XtGetGC (new, valueMask, &values);
    redraw = TRUE;
  }
  /*
   * If the indicator position has changed, or if the min/max
   * values have changed, recompute the indicator coordinates.
   */
  if (new->dial.position != current->dial.position ||
      new->dial.minimum != current->dial.minimum ||
      new->dial.maximum != current->dial.maximum){
    calculate_indicator_pos(new);
    redraw_indicator = TRUE;
  }
  /*
   * If only the indicator needs to be redrawn and
   * the widget is realized, erase the current indicator
   * and draw the new one.
   */
  if(redraw_indicator && ! redraw &&
     XtIsRealized(new) && new->core.visible){
    XDrawLine(XtDisplay(current), XtWindow(current),
              current->dial.inverse_GC,
              current->dial.center_x,
              current->dial.center_y,
             current->dial.indicator_x,
              current->dial.indicator_y);
    XDrawLine(XtDisplay(new), XtWindow(new),
              new->dial.indicator_GC,
              new->dial.center_x,
              new->dial.center_y,
              new->dial.indicator_x,
              new->dial.indicator_y);
      }
  return (redraw);
}

static void select_dial (w, event, args, n_args)
     XsDialWidget   w;
     XEvent        *event;
     char          *args[];
     int            n_args;
{
  Position   pos;
  double     angle;
  xsdialCallbackStruct cb;

  pos = w->dial.position;
  if(event->type == ButtonPress ||
        event->type == MotionNotify){
    /*
     * Get the angle in radians.
     */
   angle=atan2((double)(event->xbutton.y - w->dial.center_y),
               (double)(event->xbutton.x - w->dial.center_x));
   /*
    * Convert to degrees from the MIN_ANGLE.
```



```
    */
   angle = DEGREES(angle) - (MIN_ANGLE - 90.0);
   if (angle < 0)
     angle = 360.0 + angle;
   /*
    * Convert the angle to a position.
    */
   pos = w->dial.minimum + (angle /
            MAX_ANGLE * (w->dial.maximum - w->dial.minimum));
 }
 /*
   * Invoke the callback, report the position in the call_data
   * structure
   */
  cb.reason   = Xs_SELECTED;
  cb.event    = event;
  cb.position = pos;
  XtCallCallbacks (w, XtNselectCallback, &cb);
}
```

```
/*****************************************************************************
  * Dial.h: Public header file for Dial Widget Class
  *         From:
  *                   The X Window System,
  *              Programming and Applications with Xt
  *                   OSF/Motif Edition
  *         by
  *                Douglas Young
  *              Prentice Hall, 1990
  *
  *                   Example described on pages:  344-362
  *
  *
  *  Copyright 1989 by Prentice Hall
  *  All Rights Reserved
  *
  * This code is based on the OSF/Motif widget set and the X Window System
  *
  * Permission to use, copy, modify, and distribute this software for
  * any purpose and without fee is hereby granted, provided that the above
  * copyright notice appear in all copies and that both the copyright notice
  * and this permission notice appear in supporting documentation.
  *
  * Prentice Hall and the author disclaim all warranties with regard to
  * this software, including all implied warranties of merchantability and fitne
  * In no event shall Prentice Hall or the author be liable for any special,
  * indirect or cosequential damages or any damages whatsoever resulting from
  * loss of use, data or profits, whether in an action of contract, negligence
  * or other tortious action, arising out of or in connection with the use
  * or performance of this software.
  *
  * Open Software Foundation is a trademark of The Open Software Foundation, Inc
  * OSF is a trademark of Open Software Foundation, Inc.
  * OSF/Motif is a trademark of Open Software Foundation, Inc.
  * Motif is a trademark of Open Software Foundation, Inc.
  * DEC is a registered trademark of Digital Equipment Corporation
  * HP is a registered trademark of the Hewlett Packard Company
  * DIGITAL is a registered trademark of Digital Equipment Corporation
  * X Window System is a trademark of the Massachusetts Institute of Technology
  *****************************************************************************/

#ifndef  DIAL_H
#define  DIAL_H

extern WidgetClass XsdialWidgetClass;
typedef struct _XsDialClassRec * XsDialWidgetClass;
typedef struct _XsDialRec      * XsDialWidget;
/*
 * Define resource strings for the Dial widget.
 */
#define XtNselectCallback "selectCallback"
#define XtNmarkers        "markers"
#define XtNminimum        "minimum"
#define XtNmaximum        "maximum"
#define XtNindicatorColor "indicatorColor"
#define XtNposition       "position"
```



```
#define XtNmarkerLength   "markerLength"

#define XtCMarkers        "Markers"
#define XtCMin            "Min"
#define XtCMax            "Max"

#define Xs_SELEC        TED       1

typedef struct {
  int     reason;
  XEvent *event;
  int     position;
}  xsdialCallbackStruct;

#define Xs_SELECTED  1

#endif DIAL_H
```

```
/*****************************************************************************
 * DialP.h: Private header file for Dial Widget Class
 *         From:
 *                  The X Window System,
 *              Programming and Applications with Xt
 *                  OSF/Motif Edition
 *         by
 *               Douglas Young
 *             Prentice Hall, 1990
 *
 *                  Example described on pages:  344-362
 *
 *
 *  Copyright 1989 by Prentice Hall
 *  All Rights Reserved
 *
 * This code is based on the OSF/Motif widget set and the X Window System
 *
 * Permission to use, copy, modify, and distribute this software for
 * any purpose and without fee is hereby granted, provided that the above
 * copyright notice appear in all copies and that both the copyright notice
 * and this permission notice appear in supporting documentation.
 *
 * Prentice Hall and the author disclaim all warranties with regard to
 * this software, including all implied warranties of merchantability and fitne
 * In no event shall Prentice Hall or the author be liable for any special,
 * indirect or cosequential damages or any damages whatsoever resulting from
 * loss of use, data or profits, whether in an action of contract, negligence
 * or other tortious action, arising out of or in connection with the use
 * or performance of this software.
 *
 * Open Software Foundation is a trademark of The Open Software Foundation, Inc
 * OSF is a trademark of Open Software Foundation, Inc.
 * OSF/Motif is a trademark of Open Software Foundation, Inc.
 * Motif is a trademark of Open Software Foundation, Inc.
 * DEC is a registered trademark of Digital Equipment Corporation
 * HP is a registered trademark of the Hewlett Packard Company
 * DIGITAL is a registered trademark of Digital Equipment Corporation
 * X Window System is a trademark of the Massachusetts Institute of Technology
 *****************************************************************************/


#ifndef DIALP_H
#define DIALP_H

#define MAXSEGMENTS 200

typedef struct _XsDialClassPart{
       int ignore;
} XsDialClassPart;

typedef struct _XsDialClassRec{
   CoreClassPart    core_class;
   XsDialClassPart  dial_class;
} XsDialClassRec;
```



```
extern XsDialClassRec XsdialClassRec;


typedef struct _XsDialPart {
   Pixel     indicator_color;  /* Color of the           */
   Pixel     foreground;       /*  indicator and markers */
   int       minimum;          /* minimum value          */
   int       maximum;          /* maximum value          */
   int       markers;          /* number of marks        */
   Dimension marker_length;    /* in pixels              */
   Position  position;         /* indicator position     */
   Position  indicator_x;      /* x,y position of tip    */
   Position  indicator_y;      /*     of the indicator   */
   Position  center_x;         /* coordinates of the     */
   Position  center_y;         /*     dial center        */
   Position  inner_diam;       /* inside of markers      */
   Position  outer_diam;       /* outside of markers     */
   GC        dial_GC;          /* assorted gc's          */
   GC        indicator_GC;
   GC        inverse_GC;
   XPoint    segments[MAXSEGMENTS];
   XtCallbackList select;    /* callback list          */
} XsDialPart;

typedef struct _XsDialRec {
   CorePart          core;
   XsDialPart        dial;
} XsDialRec;

#endif DIALP_H
```